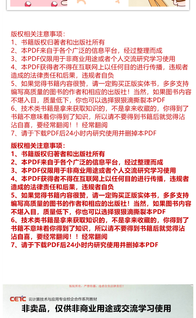


版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

CETC 云计算技术与应用专业校企合作系列教材

云存储技术与应用

主　编　朱晓彦　顾旭峰
副主编　虞　芬　龙　翔　李永亮
主　审　王　磊　刘文娟

高等教育出版社

CETC 云计算技术与应用专业校企合作系列教材

云存储技术与应用

主　编　朱晓彦　顾旭峰
副主编　虞　芬　龙　翔
　　　　李永亮
主　审　王　磊　刘文娟

高等教育出版社·北京

内容提要

本书共分 6 个单元，从最基本的磁盘分区、格式化到组建 RAID 磁盘阵列、LVM 卷，从 NFS、CIFS、iSCSI 共享到 Cinder 块存储和 Swift 对象存储，从搭建 GlusterFS 和 Ceph 分布式存储系统到使用 Ceph 和 OpenStack 进行整合、替换 OpenStack 的 Glance 和 Nova 后端存储，层层递进地讲述基本存储的概念、使用方法、主流分布式存储系统的搭建以及与云平台的整合，目的是让读者快速入门基本的云存储技术。

本书可作为高职高专院校云计算技术与应用专业和计算机网络技术专业的基础核心课程教材，也可作为云计算应用和移动应用开发技术入门的培训教材，还可作为云计算运维人员和计算机爱好者的自学用书。

图书在版编目（CIP）数据

云存储技术与应用／朱晓彦，顾旭峰主编．--北京：
高等教育出版社，2018.3
ISBN 978-7-04-049103-6

Ⅰ．①云…　Ⅱ．①朱…　②顾…　Ⅲ．①计算机网络-信息存贮-高等职业教育-教材　Ⅳ．①TP393.071

中国版本图书馆 CIP 数据核字（2017）第 313228 号

策划编辑　吴鸣飞　　责任编辑　刘子峰　　封面设计　姜　磊　　版式设计　童　丹
插图绘制　于　博　　责任校对　吕红颖　　责任印制　刘思涵

出版发行　高等教育出版社
社　　址　北京市西城区德外大街 4 号
邮政编码　100120
印　　刷　河北鹏盛贤印刷有限公司
开　　本　787 mm×1092 mm　1/16
印　　张　12
字　　数　270 千字
购书热线　010-58581118
咨询电话　400-810-0598

网　　址　http://www.hep.edu.cn
　　　　　http://www.hep.com.cn
网上订购　http://www.hepmall.com.cn
　　　　　http://www.hepmall.com
　　　　　http://www.hepmall.cn
版　　次　2018年 3 月第 1 版
印　　次　2018年 3 月第 1 次印刷
定　　价　24.80 元

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换

物 料 号　49103-00

版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

前　　言

一、缘起

在科技飞速发展的今天，人们每天所接触的信息数量也在成倍地增长。随着信息量的增加，人们无法在短时间内储存这么多的信息，因此相应的存储技术应运而生。从单一的硬盘存储到由多块硬盘组成的磁盘阵列（RAID），从传统的磁盘分区到磁盘的逻辑卷管理（LVM），从本地存储到网络存储（NFS，CIFS，iSCSI），传统的存储技术都发生了日新月异的变化。

不久前，IDC（权威市场调研机构）和 EMC（美国易安信公司）联合发布了一份名为《2020 年的数字宇宙》的预测报告，对从现在开始的 8 年内的大数据发展状况进行了预估。到 2020 年，数字宇宙将会膨胀到 40000 EB（也就是说，2020 年每个男人、女人和孩子可以均摊到 5200 GB 以上）。从现在起到 2020 年，这个数字宇宙的膨胀效果大约是每两年翻一番，而本地的存储容量将无法随着数据量的急剧增长速度而增长，这时云存储优势就得到了展现。云存储后端一般使用大规模的集群存储系统，包括 GlusterFS 存储资源池和 Ceph 分布式文件系统等。这两种存储技术都具有很好的可拓展性，可以轻松地扩展到 PB 级别且易于管理。

学习云存储技术是一个系统的过程。本书从学习和研究存储软件技术的角度展开，首先讲解本地存储技术，使读者掌握底层的存储文件系统的管理和使用；然后介绍网络存储的原理以及应用范围；最后通过构建简单的私有云存储和集群存储与私有云的对接完成一个综合项目。

二、结构

本书采用模块化的编写思路，将基础存储技术、私有云、集群存储 3 大模块划分为内置存储系统、外置存储系统、OpenStack 私有云平台构建、GlusterFS 分布式文件系统构建、Ceph 分布式文件系统构建和 OpenStack 对接 Ceph 存储 6 个教学单元，其中包含 11 个教学任务。

每个单元通过学习情境引出单元的教学核心内容，明确教学任务。每个任务的编写分为任务描述、知识学习、任务实施、项目实训 4 个环节。

任务描述：简述任务目标，展示任务实施效果，以提高学生的学习兴趣。

知识学习：详细讲解知识点，通过系列实例实践，边学边做。

任务实施：通过任务综合应用所学知识，提高学生系统运用知识的能力。

项目实训：在项目实施的基础上通过“学、仿、做”达到理论与实践的统一、知识内化的教学目的。

最后进行单元小结，总结本单元的教学重点、难点。

三、特点

1. 针对性强，教材内容选取以实用为主

本书以云计算技术专业学生的就业岗位群为导向，整个课程分为知识学习和技术应用两大部分。知识学习以云存储概述、存储技术的基本技术、OpenStack 平台的简介等基础知识为主，培养学生较为系统的云存储基本技能；技术应用以云存储的基本运用、OpenStack 平台搭建与使用、GlusterFS 和 Ceph 集群存储的构建为主要内容进行项目实训，内容设计比较丰富，便于学生理解和掌握。

2. 精心设计，教学内容与数字化资源有机结合

本书以云存储教学内容为主线将各项数字化资源有机结合在一起，形成完整的数字课程。

数字化资源包括 3 方面内容：首先，课程本身的基本信息，包括课程简介、学习指南、课程标准、整体设计、单元设计、考核方式等；其次，教学内容中重难点的微课视频教学资源，既方便课内教学，又方便学生课外预习和复习；最后，课程拓展资源，包括课程的重难点剖析、循序渐进的综合项目开发、相关培训、认证、案例、素材资源等。

四、使用

1. 教学内容课时安排

本书建议授课 64 学时，教学单元与课时安排见表 1。

表 1　教学单元与课时安排

序　号	单元名称	课时安排
1	构建内置存储系统	8
2	构建外置存储系统	8
3	构建 OpenStack 云存储系统	12
4	构建 GlusterFS 的分布式文件系统	12
5	构建 Ceph 分布式存储系统	12
6	构建超融合基础架构	12
课时总计		64

2. 课程资源一览表

本书是云计算技术与应用专业校企合作系列教材，开发了丰富的数字化教学资源，可使用的教学资源见表 2。

表 2　课程教学资源一览表

序　号	资源名称	表现形式与内涵
1	课程简介	Word 文档，包括对课程内容的简单介绍，对课时、适用对象等项目的介绍，让读者对云存储有个简单的认识
2	学习指南	Word 文档，包括对学前要求、学习目标要求以及学习路径和考核标准要求，让读者知道如何使用资源完成学习
3	课程标准	Word 文档，包括课程定位、课程目标要求以及课程内容与要求，可供教师类读者备课时使用
4	整体设计	Word 文档，包括课程设计思路，课程具体的目标要求以及课程内容设计和能力训练设计，同时给出考核方案设计，帮助教师类读者理解课程的设计理念，有助于教学实施
5	授课 PPT 和视频	PPT 文件和 MP4 视频文件，可辅助教师类读者完成教学任务
6	教学单元设计	Word 文档，分任务给出课程教案，帮助教师类读者完成每堂课的教学细节分析
7	微课	MP4 视频文件，提供给读者更加直观的学习，有助于学习知识
8	电子课件	PPT 文件，提供 PowerPoint 2003 版使用，也可根据实际需要加以修改后使用
9	实训任务单	Word 文档，为每个任务设计实训来加深课堂知识的学习，并给出实训的详细步骤
10	案例	Tar 包，包括单元项目案例和综合案例，综合运用所学知识
11	习题库、试卷库	Word 文档，习题包括理论习题和操作习题，试卷包括单元测试和课程测试。通过练习和测试，让读者加深对知识的掌握程度
12	附书源码	Tar 包，包括本书中所有例题和任务的源代码

续表

本书配套的资源包、运行脚本、电子教案等，读者可登录 http://www.1daoyun.com 下载。

五、致谢

本书由南京第五十五所技术开发有限公司、江苏一道云科技发展有限公司共同编写，朱晓彦、顾旭峰任主编。在此对各位参编老师、工程师致以衷心的感谢。

由于作者水平有限，错误和不足之处在所难免，恳请各位读者给予批评、指正，将不胜感激。

编　者
2017 年 10 月

版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

目　录

单元 1
构建内置存储系统

学习目标

【知识目标】

- 了解内置存储的分区、格式化、挂载等操作。
- 了解 RAID 的组建和使用环境。
- 了解 LVM 卷的组建和使用。

PPT1：构建内置存储系统

【技能目标】

- 掌握硬盘的分区、格式化、使用的具体操作步骤。
- 掌握 RAID 各个等级的组建、使用的具体操作步骤。
- 掌握 LVM 卷的组建、使用的具体操作步骤。

学习情境

某公司研发部门随着规模不断扩大，研发人员不断增多，在对平常办公研发总结中，发现公司的部门之间存在消息传递不及时、消息滞后、各种研发资料和部署手册版本不统一等弊端，这给部门的正常运转造成了不利的影响。公司研发部了解到可以通过统一存储将每个人的资料进行统一管理，每种资料进行版本划分，于是安排工程师小缪对统一存储进行研究。在准备这个项目时，小缪首先对内置存储系统的构建进行调研和安装测试，并按照以下要求进行环境规划和准备。

1. 项目设计

文件服务器一台，配置为双千兆网卡，一块系统盘，4 块 1 TB 硬盘，双核 CPU。

2. 实现存储业务基本规划

（1）RAID 规划

2 个分区做 RAID 0，2 个分区做 RAID 1，3 个分区做 RAID 5。

（2）分区与格式化的基本操作

① RAID 基础运维。

② LVM 逻辑卷基础运维。

（3）服务器功能实现

① 对硬盘进行分区，组建 RAID，挂载使用。

② 对硬盘进行分区，组建 LVM 卷，挂载使用。

任务 1.1　学习系统分区和文件系统

任务描述

1. 学会使用 VirtualBox 安装 CentOS 7 虚拟机。
2. 了解分区，并学会使用分区工具进行分区。
3. 了解 Linux 文件系统，并会使用文件系统进行格式化。

知识学习

1. Fdisk 分区

（1）分区的定义

分区是将一个硬盘驱动器分成若干个逻辑驱动器，把硬盘连续的区块当作一个独立的磁盘使用。分区表是一个硬盘分区的索引，分区的信息都会写进分区表。

笔记

（2）分区的作用

① 防止数据丢失：如果系统只有一个分区，如果这个分区损坏，用户将会丢失所有的数据。

② 增加磁盘空间使用效率：可以用不同的区块大小来格式化分区，如果有很多大小为 1 KB 的文件，而硬盘分区区块大小为 4 KB，那么每存储一个文件将会浪费 3 KB 空间。这时需要取这些文件大小的平均值进行区块大小的划分。

③ 数据激增到极限不会引起系统挂起：将用户数据和系统数据分开，可以避免用户数据填满整个硬盘引起的系统挂起。

（3）分区工具 Fdisk

Fdisk 命令参数如下：

① p——打印分区表。

② n——新建一个新分区。

③ d——删除一个分区。

④ q——退出不保存。

⑤ w——把分区写进分区表，保存并退出。

2. 文件系统

Linux 文件系统中的文件是数据的集合，文件系统不仅包含着文件中的数据，还包括文件系统的结构，所有 Linux 用户和程序所看到的文件、目录、软连接及文件保护信息等都存储在其中。以下是几种主流的文件系统。

（1）Ext 3

Ext 3（Third Extended Filesystem）是第 3 代扩展文件系统，文件系统存储单位是“块”，其相当于 NTFS 中的“簇”。格式化硬盘或分区时将所有磁盘空间分成若干个

大小相同块。块的大小可以在格式化时指定，也可以采用默认设置。每个块都有一个唯一编号，从0开始。0号块起始于文件系统的起始扇区。

Ext 3文件系统将若干个块组成“块组”，每个块组大小相同。但由于块的总数不一定是块组的整倍数，所以最后一个块组相对于其他块组要小。每个块组都对应一个块组描述符，这些块组描述符统一放在文件系统的前面，对块组进行管理。

Ext 3文件系统使用“i节点”来记录文件的时间、大小、块指针等信息；用目录项描述文件名和节点号，通过节点号就能访问其节点信息。

（2）Ext 4

Ext 4（Fourth Extended Filesystem）是第四代扩展文件系统，是Linux系统下的日志文件系统，是Ext 3文件系统的后继版本，是由Ext 3的维护者Theodore Tso领导的开发团队实现的，并引入到Linux 2.6.19内核中。

Ext 4的产生原因是开发人员在Ext 3中加入了新的高级功能，但在实现的过程出现了以下几个重要问题：

① 一些新功能违背向后兼容性。

② 新功能使Ext 3代码变得更加复杂并难以维护。

③ 新加入的更改使原来十分可靠的Ext 3变得不可靠。

由于这些原因，从2006年6月开始，开发人员决定把Ext 4从Ext 3中分离出来进行独立开发。Ext 4的开发工作从那时起开始进行，但大部分Linux用户和管理员都没有太关注这件事情，直到Linux 2.6.19内核在2006年11月发布。Ext 4第一次出现在主流内核里，但当时还处于试验阶段，因此很多人都忽视了它。

2008年12月25日，Linux Kernel 2.6.28的正式版本发布。随着这一新内核的发布，Ext 4文件系统也结束实验期，成为稳定版。

（3）XFS

笔记

XFS文件系统是SGI开发的高级日志文件系统，极具伸缩性，非常健壮。在Linux环境下，目前可用的最新XFS文件系统版本为1.2，可以很好地工作在Linux 2.4核心下。

其具有以下主要特性：

① 数据完全性。采用XFS文件系统，当意想不到的宕机发生后，首先，由于文件系统开启了日志功能，所以用户磁盘上的文件不会因意外宕机而遭到破坏。不论目前文件系统上存储的文件与数据有多少，文件系统都可以根据所记录的日志在很短的时间内迅速恢复磁盘文件内容。

② 传输特性。XFS文件系统采用优化算法，日志记录对整体文件操作影响非常小。XFS查询与分配存储空间非常快。XFS文件系统能连续提供快速的反应时间。根据对XFS、Ext 4、Ext 3、Btrfs等文件系统的测试，XFS文件系统的性能表现相当出众。

③ 可扩展性。XFS是一个全64位的文件系统，它可以支持上百万TB的存储空间，对特大文件及小尺寸文件的支持都表现出众，支持特大数量的目录。最大可支持的文件大小为9 EB，最大文件系统尺寸为18 EB。XFS使用高的表结构（B+树），保证文件系统可以快速搜索与空间分配。XFS能够持续提供高速操作，文件系统的性能不受目录中目录及文件数量的限制。

④ 传输带宽。XFS 能以接近裸设备 I/O 的性能存储数据。在单个文件系统的测试中，其吞吐量最高可达 7 GB/s；对单个文件的读写操作，其吞吐量可达 4 GB/s。

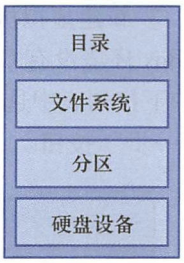


图 1-1　分区与文件系统的结构关系

3. 分区与文件系统的关系

硬盘设备是最底层的，在硬盘之上可以创建分区，对于创建的分区可以格式化为相应的文件系统，然后挂载使用。分区与文件系统的结构关系如图 1-1 所示。

任务实施

1. 配置系统基本环境

本次实验的环境为 CentOS 7 系统，在 Windows 中可以通过 VirtualBox 安装 CentOS 7 虚拟机进行实验。

① 安装 VirtualBox。VirtualBox 的安装很简单，可一直按默认设置进行安装，不断地单击“下一步”按钮即可。中间如果遇到需要安装的驱动，选择安装。CentOS 7 的安装源文件为 CentOS-7-x86_64-DVD-1511. iso。

② 创建 CentOS 7 虚拟机。打开 VirtualBox，如图 1-2 所示。单击左上方“新建”按钮，创建 CentOS 7 环境。

图 1-2　VirtualBox 主界面

③ 新建虚拟机环境。虚拟机（系统中称为虚拟电脑）名称为 Centos7，由于 VirtualBox 环境没有 CentOS 的选项，所以在“版本”选项中单击右侧的下拉按钮，在弹出的下拉列表中选择“Red Hat（64 Bit）”选项，如图 1-3 所示。完成设置后单击“下一步”按钮。

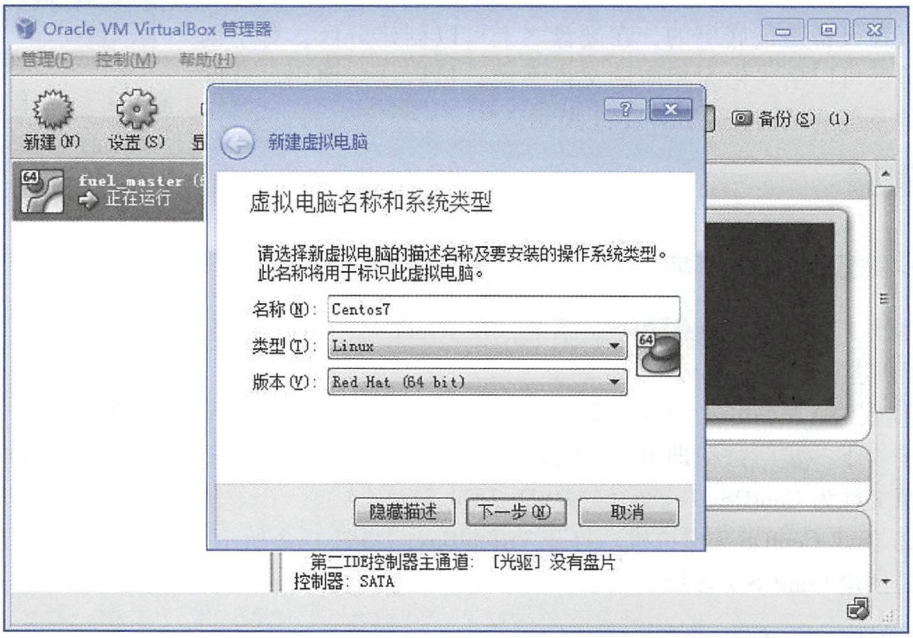


图 1-3 新建虚拟电脑

④ 分配内存大小。动态分配内存大小，如图 1-4 所示。这时用户可以拖动“滑块”来设置内存大小，建议内存空间设置大一点。完成后，单击“下一步”按钮。

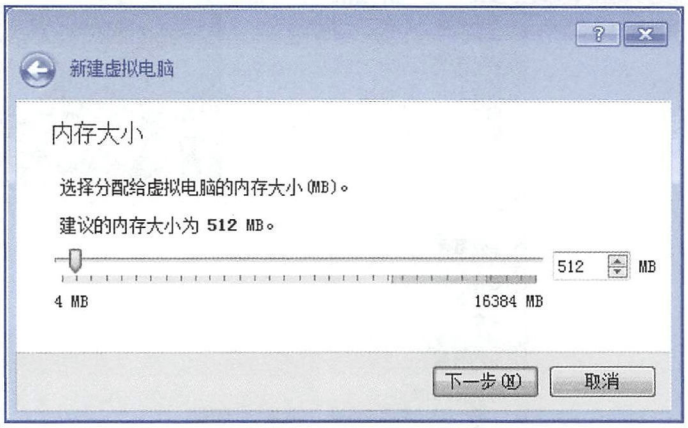


图 1-4 分配内存大小

⑤ 选择虚拟硬盘文件类型。选中“VDI（VirtualBox，磁盘映像）”单选按钮，设置虚拟硬盘文件类型为 VDI，如图 1-5 所示。完成后，单击“下一步”按钮。

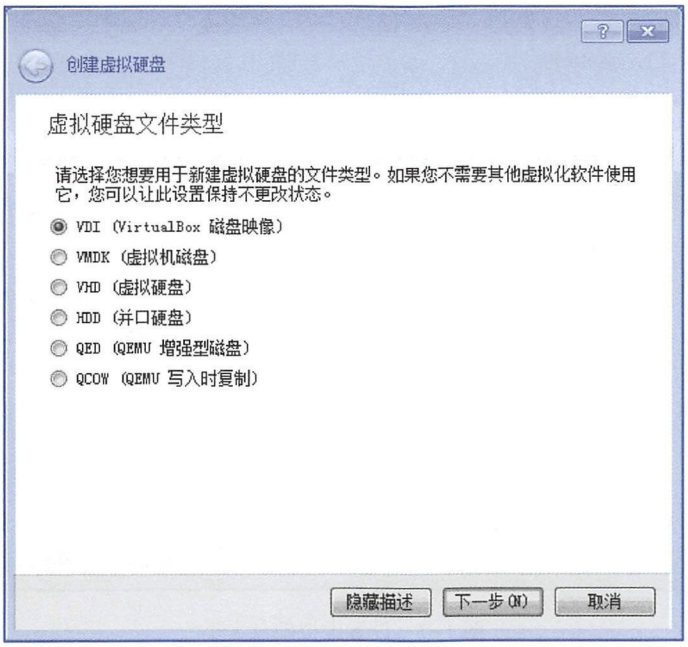


图 1-5　创建虚拟硬盘文件类型

⑥ 选择存储模式。选中“固定大小”单选按钮，设置存储在物理硬盘上的方式，如图 1-6 所示。完成后，单击“下一步”按钮。

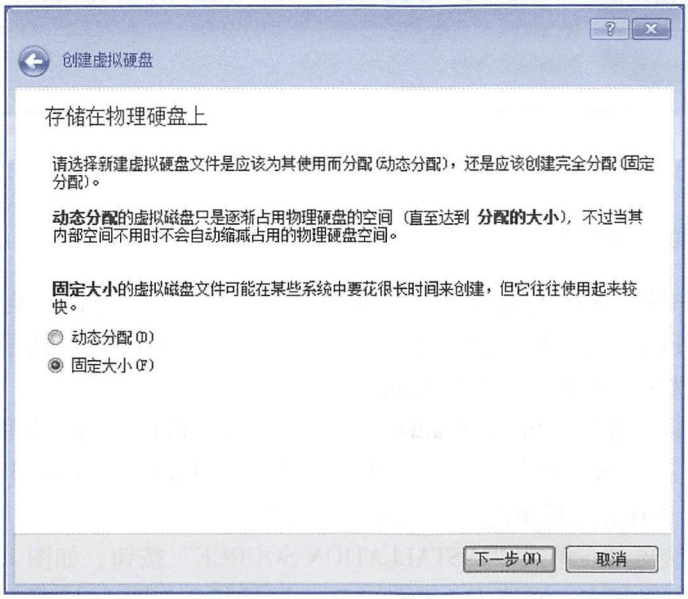


图 1-6　选择虚拟硬盘模式

⑦ 选择磁盘大小。拖动“滑块”，创建虚拟硬盘大小为 8 GB，如图 1-7 所示。然后单击“创建”按钮。

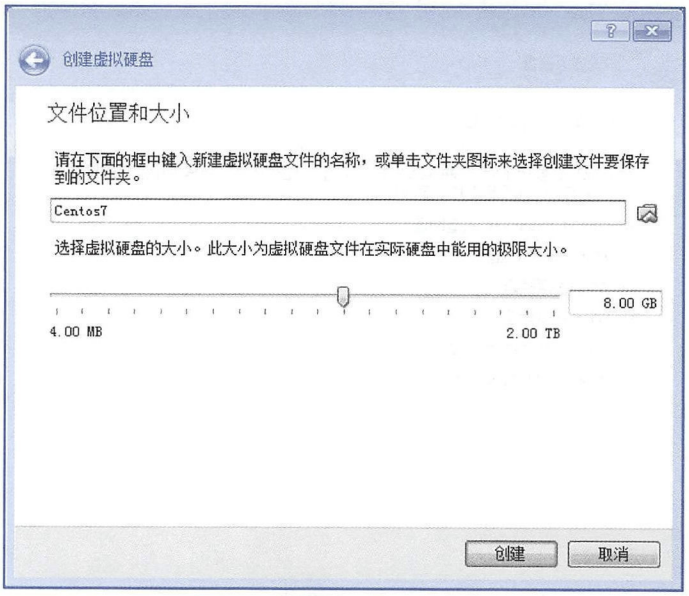


图 1-7 设置虚拟硬盘大小

⑧ 创建磁盘。出现进度条，显示正在创建虚拟硬盘的进度，如图 1-8 所示。

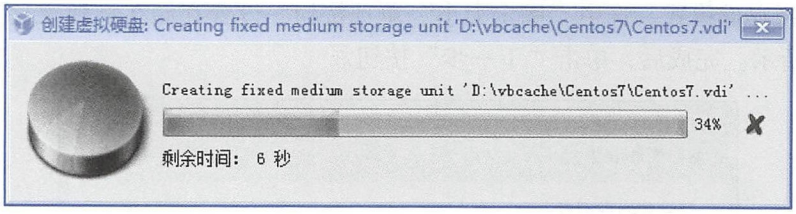


图 1-8 正在创建虚拟硬盘

笔 记

2. 安装虚拟机操作系统

① 启动虚拟机。磁盘创建完毕后，在打开的对话框中，单击“启动虚拟机”按钮，然后在打开的“选择启动盘”对话框中，选择“物理设备”选项为启动盘，如图 1-9 所示。选定后单击“启动”按钮。

② 系统安装。选择“Install CentOS 7”命令并按〈Enter〉键，如图 1-10 所示。

③ 安装语言升级。选择语言为“English”→“English（United States）”，如图 1-11 所示。选择完之后单击“Continue”按钮。

④ 选择安装磁盘。单击“INSTALLATION SOURCE”按钮，如图 1-12 所示。

⑤ 选择安装的硬盘。单击左上角的“Done”按钮，如图 1-13 所示。

⑥ 系统安装。单击“Begin Installation”按钮，如图 1-14 所示。

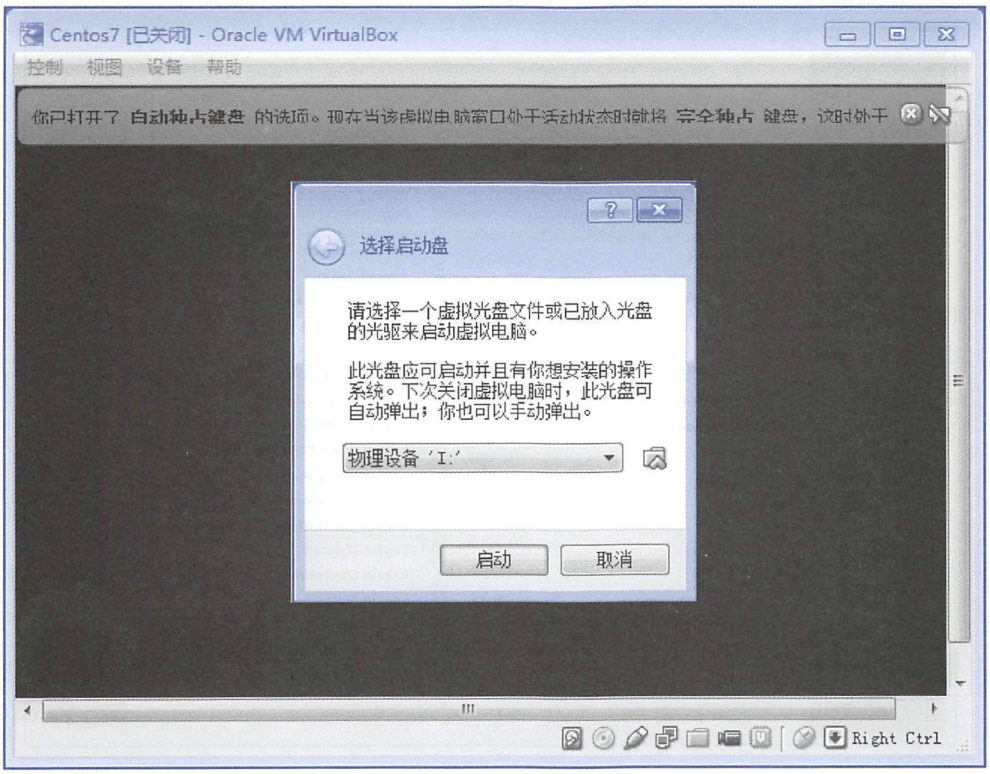


图 1-9　选择启动盘

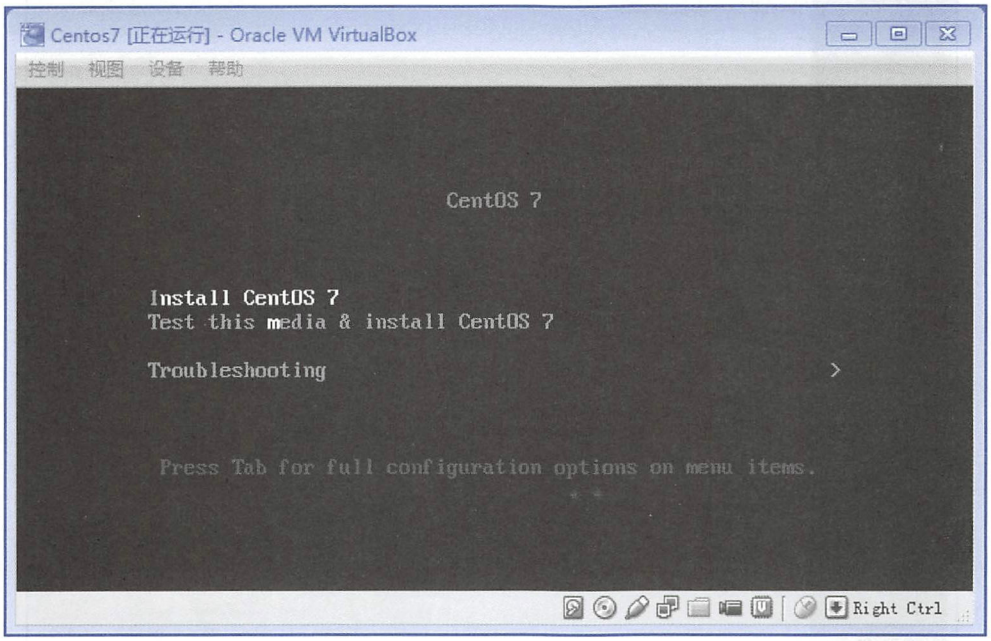


图 1-10　安装 CentOS 7

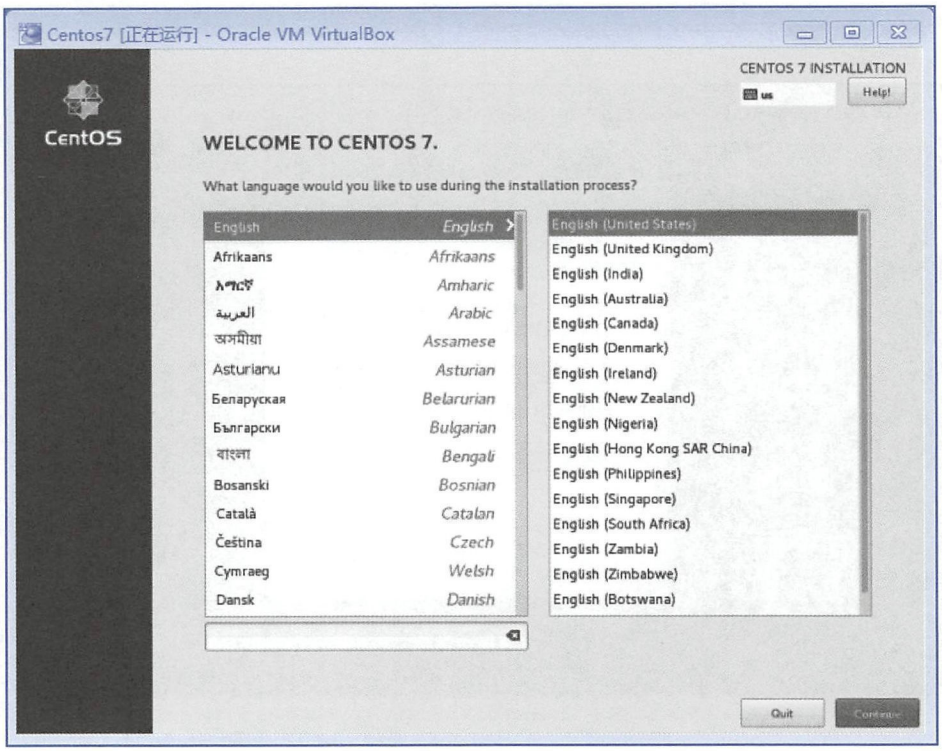


图 1-11 选择语言

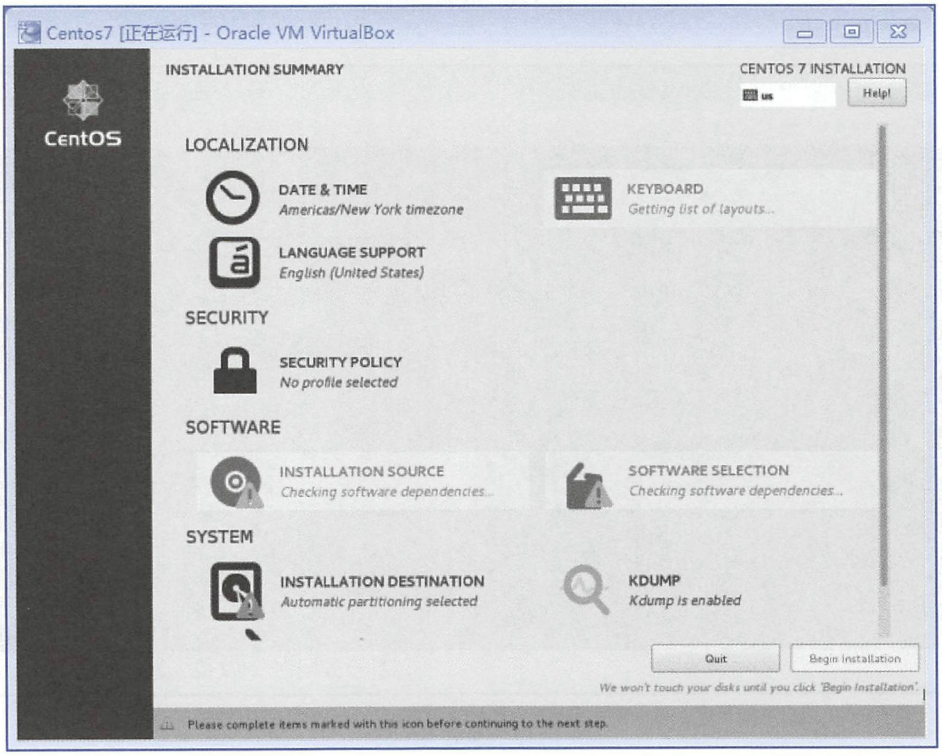


图 1-12 选择安装磁盘

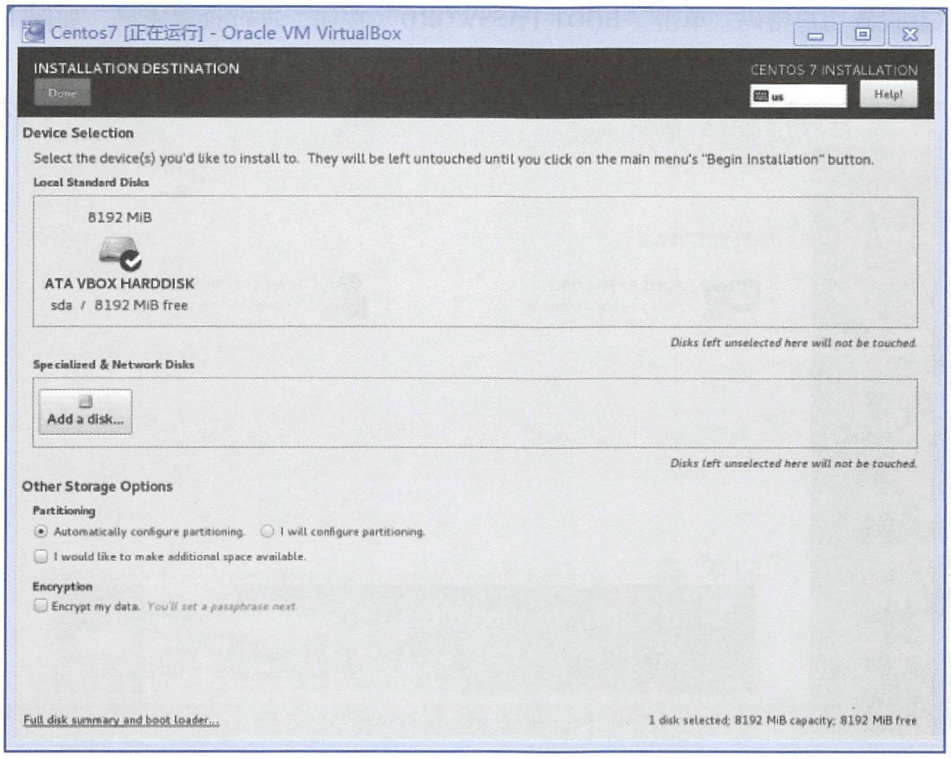


图 1-13　选择安装的硬盘

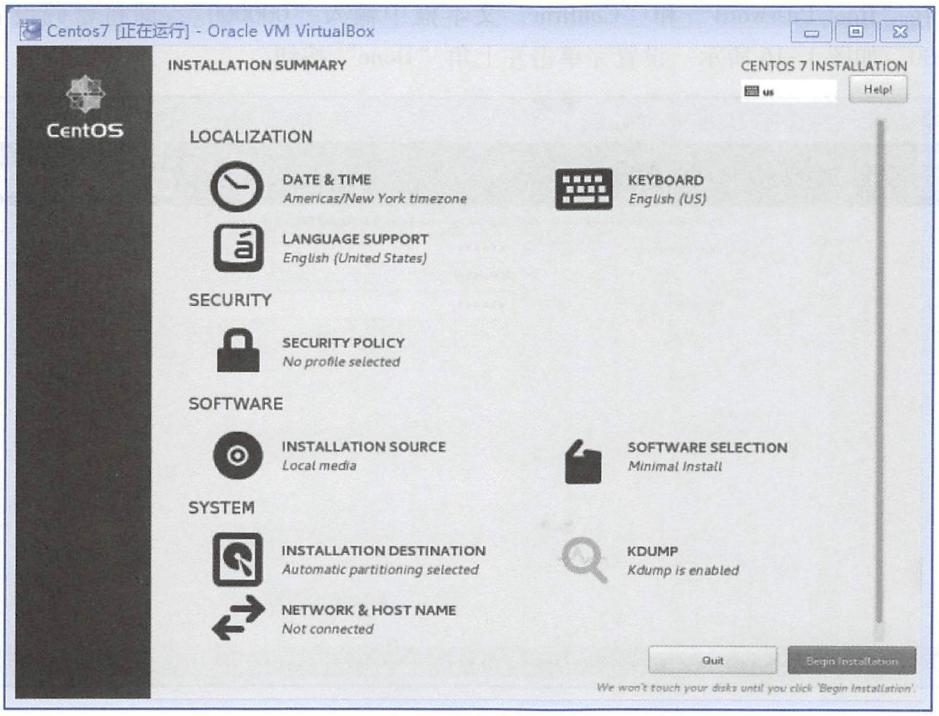


图 1-14　开始系统安装

⑦ 设置用户密码。单击“ROOT PASSWORD”按钮，进行设置密码，如图 1-15 所示。

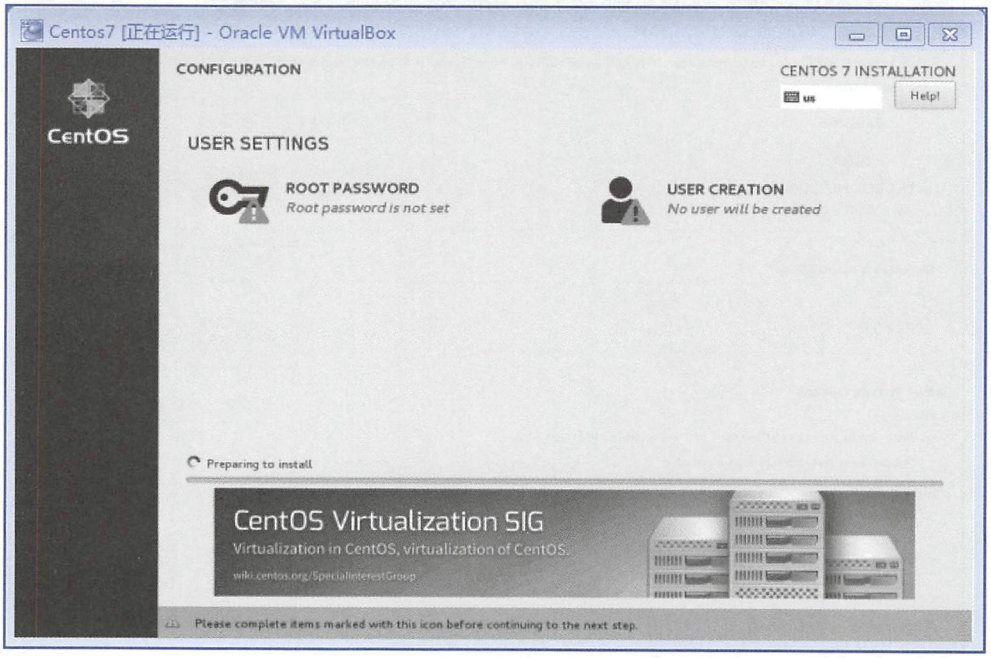


图 1-15 设置密码

在“Root Password”和“Confirm”文本框中输入“000000”，即将密码设置为 000000，如图 1-16 所示。设置完单击左上角“Done”按钮。

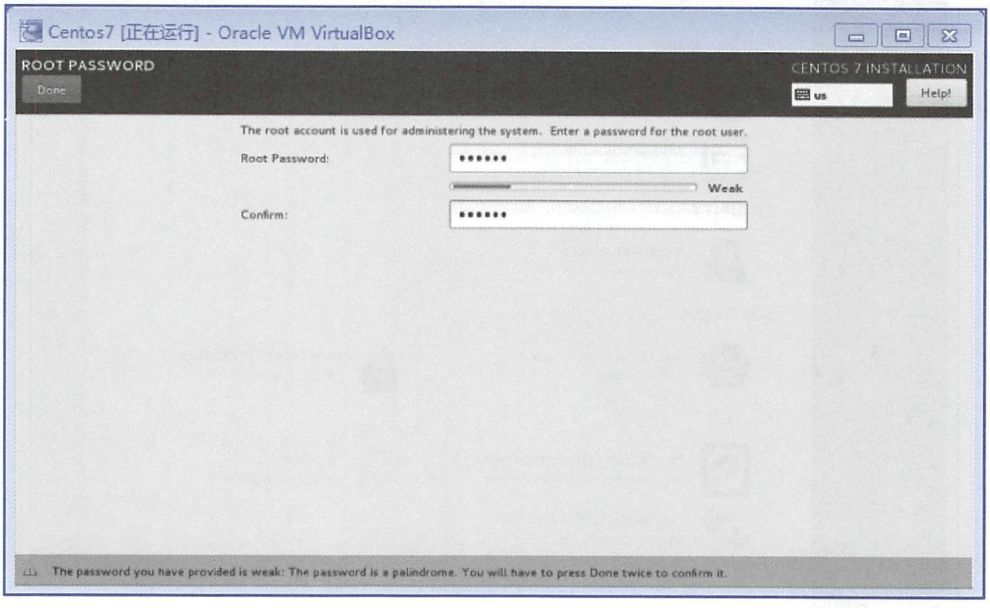


图 1-16 设置密码完成

系统安装过程如图 1-17 所示。

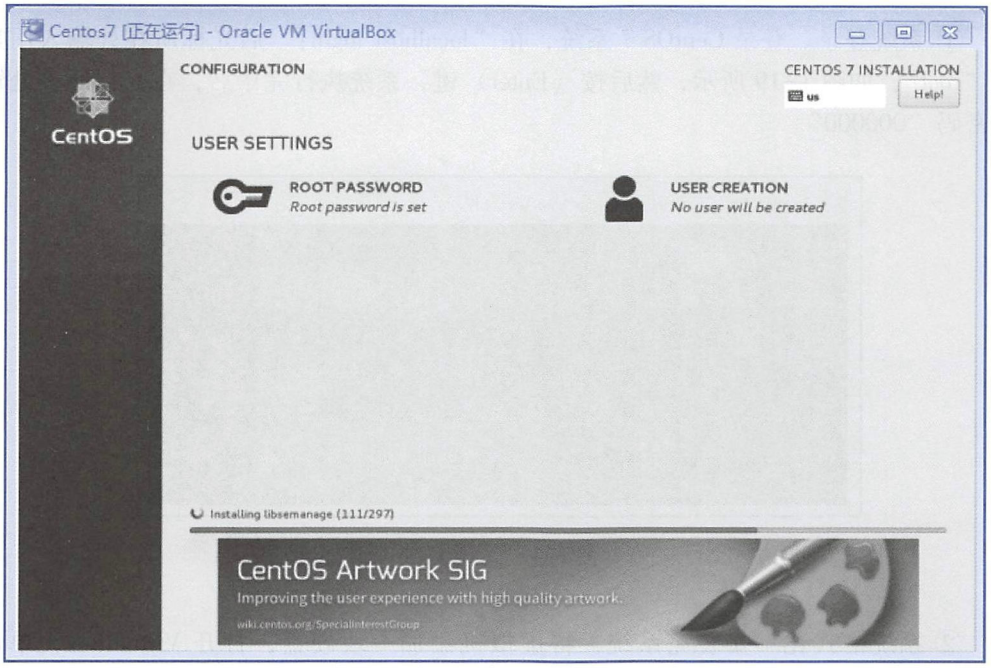


图 1-17　系统安装过程

⑧ 重启系统。安装完系统，单击“Reboot”按钮重启，如图 1-18 所示。

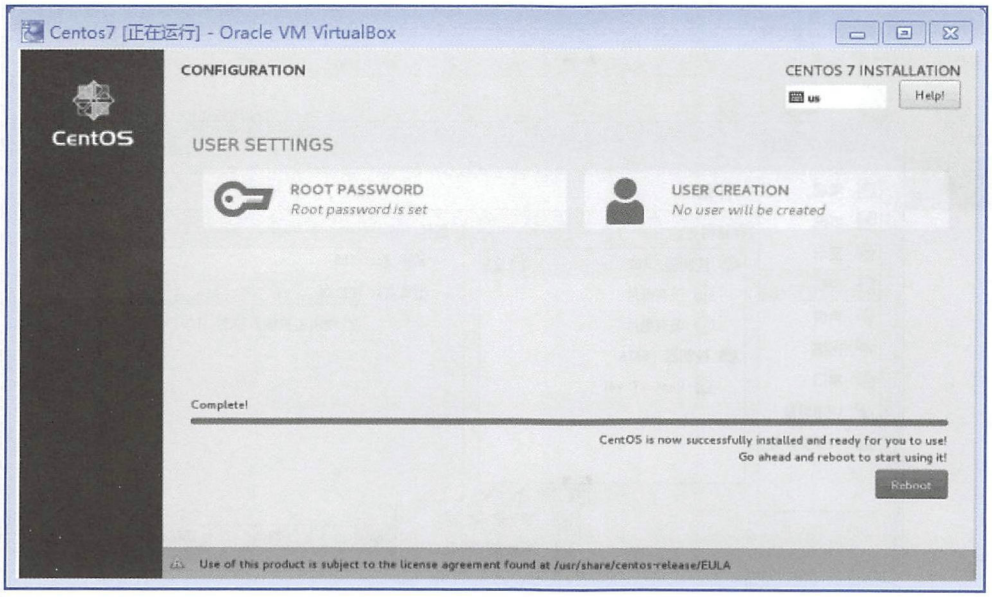


图 1-18　重启系统

3. 初始系统环境配置

① 系统登录。登录 CentOS 7 系统，在“localhost login:”后光标所在处输入用户名“root”，如图 1-19 所示，然后按〈Enter〉键，系统执行完毕后，在光标所在处输入密码“000000”。

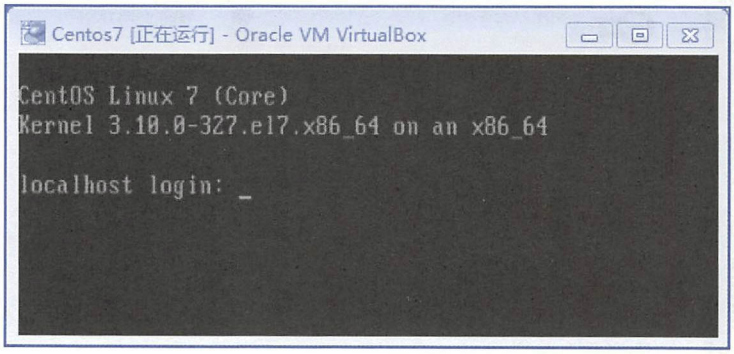


图 1-19　登录系统

② 系统格式化。安装完系统，将虚拟机添加一块硬盘，打开 VirtualBox，单击“设置”按钮，在打开的“Centos 7 设置”对话框中，选择左侧列表中的“存储”选项卡，在“储存树”列表框下单击按钮，然后在弹出的下拉菜单中选择“添加虚拟硬盘”命令，如图 1-20 所示。

图 1-20　添加虚拟硬盘

③ 新增硬盘。在打开的“虚拟电脑控制台-问题”对话框中，单击“创建新的虚拟盘”按钮，如图 1-21 所示。

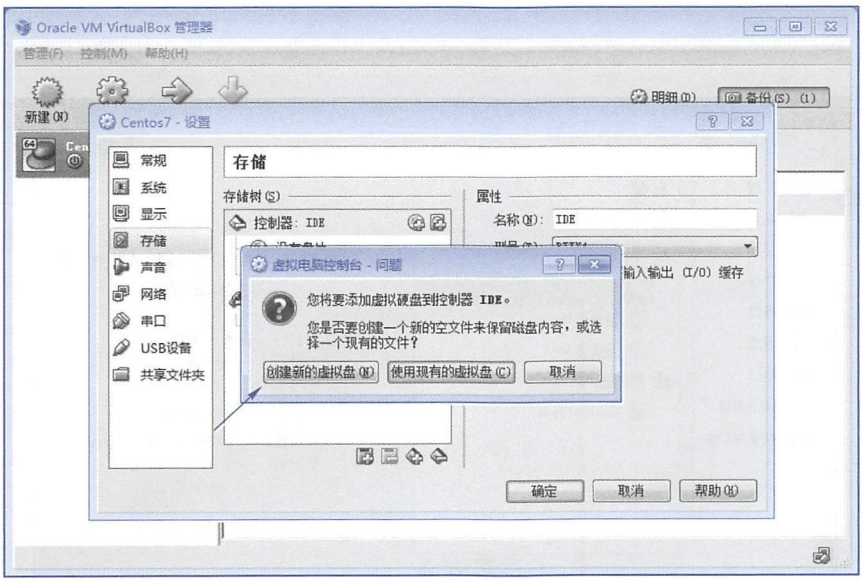


图 1-21　创建新的虚拟盘

在打开的对话框中滑动“滑块”，设置“文件大小”为 50 GB，在“虚拟硬盘文件类型”项目栏中选中“VDI（VirtualBox 磁盘映象）”单选按钮，在“存储在物理硬盘上”列表中选中“动态分配”单选按钮，如图 1-22 所示。

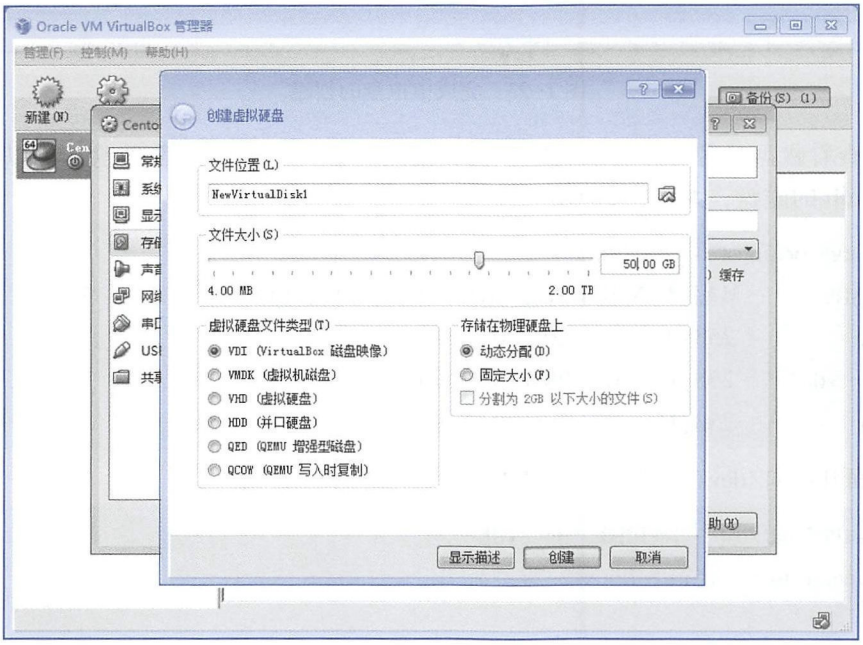


图 1-22　设置虚拟盘选项

单击“确定”按钮，返回到“Centos 7-设置”对话框中，单击“确定”按钮，如图 1-23 所示。

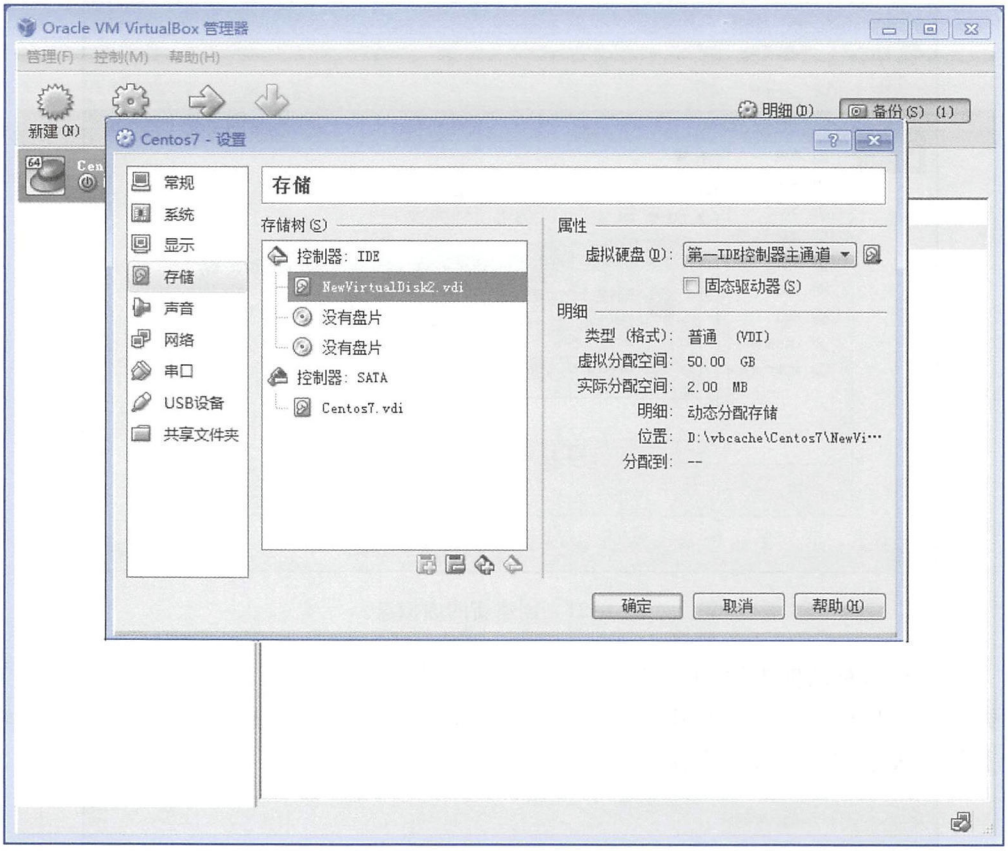


图 1-23　完成虚拟盘的创建

④ 查看磁盘。启动并登录到虚拟机，使用“lsblk”命令查看磁盘，可以看到一块名为 vdb 的磁盘，大小为 50 GB。

```
[root@localhost ~]# lsblk
NAME        MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda         253:0    0   20G  0 disk
└─vda1      253:1    0   20G  0 part   /
vdb         253:16   0   50G  0 disk
```

下面开始对/dev/vdb 进行分区。

```
[root@localhost ~]# fdisk /dev/vdb
Welcome to fdisk(util-linux 2.23.2).

Changes will remain in memory only,until you decide to write them.
```

```
Be careful before using the write command.

Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0xd62cd77d.

Command(m for help):
```

输入“p”命令显示分区表。

```
Command(m for help):p

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1 * 512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

   Device Boot      Start         End      Blocks   Id  System

Command(m for help):
```

输入“n”命令对磁盘进行分区。

```
Command(m for help):n
Partition type:
   p   primary(0 primary,0 extended,4 free)
   e   extended
Select(default p):
```

出现两个菜单项，p 代表主分区（默认选择），e 代表是扩展分区。此处选择主分区，按〈Enter〉键，出现提示“Partition number”。输入主分区号，默认选择的是 1，按〈Enter〉键，开始柱面也选择默认，直接按〈Enter〉键。创建的分区的大小为 10 GB，语法为“+10G”。

```
Select(default p):
Using default response p
Partition number(1-4,default 1):
First sector(2048-104857599,default 2048):
Using default value 2048
Last sector,+sectors or +size{K,M,G}(2048-104857599,default 104857599):+10G
```

笔记

创建好了之后输入“p”命令查看。

```
Command(m for help):p

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1*512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

Device Boot      Start         End      Blocks   Id  System
/dev/vdb1         2048    20973567    10485760   83  Linux

Command(m for help):
```

从结果中可以看到创建了一个名称为 vdb1、大小为 10 GB 的分区。
输入“w”命令保存并退出。

```
Command(m for help):w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

使用“fdisk -l”命令查看磁盘分区。

```
[root@localhost ~]# fdisk -l

Disk /dev/vda:21.5 GB,21474836480 bytes,41943040 sectors
Units=sectors of 1*512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0x000af71d

Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *     2048    41929649    20963801   83  Linux

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1*512=512 bytes
```

```
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

Device Boot      Start         End      Blocks   Id  System
/dev/vdb1        2048      20973567    10485760   83  Linux
```

接下来使用分区，先对分区进行格式化，这里使用 Ext 4 格式，还可以使用 Ext 3 或者 XFS 的格式。

```
[root@localhost ~]# mkfs. ext4 /dev/vdb1
mke2fs 1. 42. 9(28-Dec-2013)
Filesystem label=
OS type:Linux
Block size=4096(log=2)
Fragment size=4096(log=2)
Stride=0 blocks,Stripe width=0 blocks
655360 inodes,2621440 blocks
131072 blocks(5. 00%)reserved for the super user
First data block=0
Maximum filesystem blocks=2151677952
80 block groups
32768 blocks per group,32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768,98304,163840,229376,294912,819200,884736,1605632

Allocating group tables:done
Writing inode tables:done
Creating journal(32768 blocks):done
Writing superblocks and filesystem accounting information:done
```

创建挂载目录。

```
[root@localhost ~]# mkdir -p /mnt/vdb1
```

挂载分区。

```
[root@localhost ~]# mount /dev/vdb1 /mnt/db1
```

查看挂载情况。

笔记

```
[root@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        20G  977M   20G   5% /
devtmpfs        984M     0  984M   0% /dev
tmpfs          1001M     0 1001M   0% /dev/shm
tmpfs          1001M   17M  985M   2% /run
tmpfs          1001M     0 1001M   0% /sys/fs/cgroup
/dev/vdb1       9.8G   37M  9.2G   1% /mnt/vdb1
```

现在往/mnt/vdb1 目录下写的内容都保存到/dev/vdb1 中了。

删除分区也是使用“fdisk /dev/vdb”命令，然后输入“d”命令，根据提示操作选择要删除的分区就可以了。同样，输入“w”命令后保存退出。

删除刚才创建的主分区 vdb1。因为只创建了一个分区，输入“d”命令后会默认删除该分区。

首先需要卸载挂载的目录。

```
[root@localhost ~]# umount /dev/vdb1
```

然后查看挂载情况。

```
[root@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        20G  977M   20G   5% /
devtmpfs        984M     0  984M   0% /dev
tmpfs          1001M     0 1001M   0% /dev/shm
tmpfs          1001M   17M  985M   2% /run
tmpfs          1001M     0 1001M   0% /sys/fs/cgroup
```

可以看到/dev/vdb1 已经没有挂载了。

接下来进行删除分区操作。

```
[root@localhost ~]# fdisk /dev/vdb
Welcome to fdisk(util-linux 2.23.2).

Changes will remain in memory only,until you decide to write them.
Be careful before using the write command.


Command(m for help):d
Selected partition 1
Partition 1 is deleted
```

```
Command(m for help):
```

输入“p”命令查看分区。

```
Command(m for help):p

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1*512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

   Device Boot      Start         End      Blocks   Id  System

Command(m for help):
```

此时，可以看到之前创建的 vdb1 已经不见了。因为系统只有一个分区，输入“d”命令会默认删除该分区，并不会让用户选择。

输入“w”命令保存并退出。

```
Command(m for help):w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

项目实训

【实训题目】

挂载一个 30 GB 的分区 vdb1 到/mnt/vdb1 目录下。

【实训目的】

1. 掌握 Fdisk 分区工具的使用方法。
2. 掌握多种文件系统的格式化。
3. 掌握对硬盘的挂载和使用。

【实训内容】

1. 使用 Fdisk 分区工具，对/dev/vdb 进行分区，创建大小为 30 GB，名称为 vdb1 的分区。

2. 使用 Ext 4 文件系统对该分区进行格式化。

3. 在 mnt 目录下创建一个子目录 vdb1。
4. 挂载/dev/vdb1 到新创建的/mnt/vdb1。

任务 1.2 学习 RAID 磁盘阵列

任务描述

1. 了解 RAID 及其各个等级之间的区别。
2. 了解 RAID 的优势和使用场景。
3. 掌握 RAID 各种等级的组建、使用和运维。

知识学习

笔记

1. RAID 的定义

RAID（Redundant Array of Indenpensive Disk，独立磁盘冗余阵列，简称磁盘阵列）是指把多个磁盘组成一个阵列，当作单一磁盘使用，它将数据以分段或条带（Striping）的方式储存在不同的磁盘中，存取数据时，阵列中的相关磁盘一起动作，大幅减低数据的存取时间，同时有更佳的空间利用率。磁盘阵列利用不同的技术，称为 RAID Level，不同的 Level 针对不同的系统及应用，以解决数据安全的问题。简单来说，RAID 把多个硬盘组合成为一个逻辑扇区，因此，操作系统只会把它当作一个硬盘。

（1）RAID 的优点

① 提高传输速率。RAID 通过在多个磁盘上同时存储和读取数据来大幅提高存储系统的数据吞吐量（Throughput）。在 RAID 中，可以让很多磁盘驱动器同时传输数据，而这些磁盘驱动器在逻辑上又是一个磁盘驱动器，所以使用 RAID 可以达到单个磁盘驱动器几倍、几十倍甚至上百倍的速率。这也是 RAID 最初想要解决的问题。当时 CPU 的速度增长很快，而磁盘驱动器的数据传输速率无法大幅提高，需要有一种方案解决二者之间的矛盾，于是便产生了 RAID。

② 通过数据校验提供容错功能。若不包括写在磁盘上的 CRC（循环冗余校验）码，普通磁盘驱动器是无法提供容错功能的。RAID 容错是建立在每个磁盘驱动器的硬件容错功能之上的，所以它提供更高的安全性。在很多 RAID 模式中都有较为完备的相互校验/恢复的措施，甚至是直接相互的镜像备份，从而大大提升了 RAID 系统的容错度，提高了系统的稳定冗余性。

（2）RAID 的缺点

① 做不同的 RAID，存在 RAID 模式硬盘利用率低、价格昂贵的缺点。

② RAID 0 没有冗余功能，如果一个磁盘（物理）损坏，则所有的数据都无法使用。

③ RAID 1 磁盘的利用率只有 50%，是所有 RAID 级别中最低的。

④ RAID 5 可以理解为是 RAID 0 和 RAID 1 的折中方案。RAID 5 可以为系统提供数据安全保障，但保障程度要比 RAID 1 低，而磁盘空间利用率要比 RAID 1 高。

（3）RAID 的分类

① RAID 0，即数据分条（条带）盘，只需要两块以上的硬盘，成本低，可以提高整个磁盘的性能和吞吐量，但没有容错功能，任何一个磁盘的损坏都将损坏全部数据。RAID 0 结构如图 1-24 所示。

② RAID 1，磁盘镜像盘，数据在写入一块磁盘的同时，会在另一块闲置的磁盘上生成镜像文件，至少需要两块硬盘，RAID 大小等于两个 RAID 分区中最小的容量（最好将分区大小分为一样），可增加热备盘提供一定的备份能力。数据有冗余，在存储时同时写入两块硬盘，实现了数据备份，但相对降低了写入性能；读取数据时可以并发，几乎类似于 RAID 0 的读取效率。RAID 1 结构如图 1-25 所示。

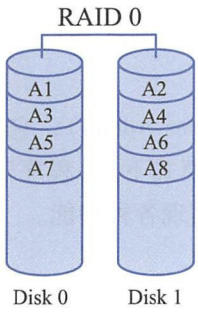


图 1-24　RAID 0 结构

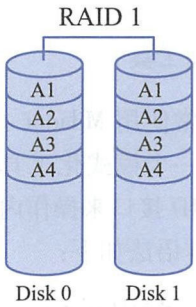


图 1-25　RAID 1 结构

③ RAID 5，即分布式奇偶校验的独立磁盘结构，需要 3 块或以上硬盘，可以提供热备盘实现故障的恢复。采用奇偶校验的可靠性强，且只有两块硬盘同时损坏时数据才会完全损坏，而只损坏一块硬盘时，系统会根据存储的奇偶校验位重建数据，临时提供服务。此时如果有热备盘，系统还会自动在热备盘上重建故障磁盘上的数据。RAID 5 结构如图 1-26 所示。

笔记

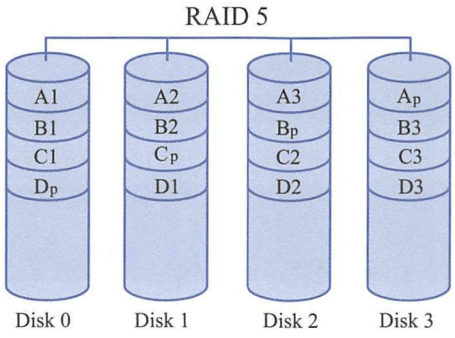


图 1-26　RAID 5 结构

④ RAID 1+0（RAID 10）是先镜射再分区数据，再将所有硬盘分为两组，视为是 RAID 0 的最低组合，然后将这两组各自视为 RAID 1 运作。RAID 1+0 结构如图 1-27 所示。

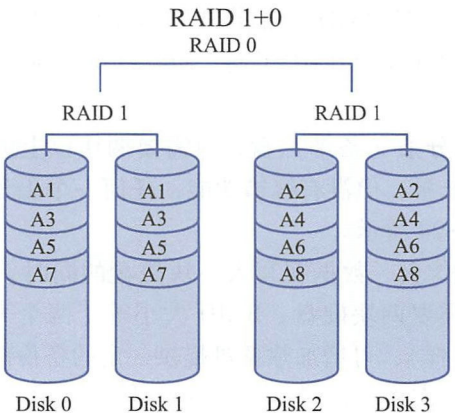


图 1-27 RAID 1+0 结构

2. RAID 的管理工具

笔记

RAID 的管理通常使用 Mdadm（Multiple Devices Admin），它是 Linux 下标准的软 RAID 管理工具，是一个模式化工具（在不同的模式下）。程序工作在内存用户程序区，为用户提供 RAID 接口来操作内核的模块，实现各种功能。

Mdadm 命令基本语法如下：

```
mdadm [mode] <RAID-device> [options] <component-devices>
```

目前支持的模式有 LINEAR（线性模式）、RAID 0（Striping 条带模式）、RAID 1（Mirroring）、RAID 4、RAID 5、RAID 6、RAID 10、MULTIPATH 和 FAULTY 等多种。

查看 RAID 阵列的详细信息：

```
mdadm -D /dev/md
--detail
```

停止 RAID 阵列：

```
mdadm -S /dev/md
--stop
```

启动 RAID 阵列：

```
mdadm -A /dev/md
--start
```

其他选项如下。

-c，--config=：指定配置文件，默认为/etc/mdadm. conf。

-s，--scan：扫描配置文件或用/proc/mdstat 以搜寻丢失的信息。默认配置文件为/etc/mdadm. conf。

-h，--help：帮助信息，用在以上选项后，则显示该选项信息。

-v，--verbose：显示细节，一般只能跟--detail 或--examine 一起使用，显示中级的信息。

-b，--brief：较少的细节。用于--detail 和--examine 选项。

--help-options：显示更详细的帮助。

-V，--version：版本信息。

-q，--quiet：安静模式；加上该选项能使 Mdadm 不显示纯消息性的信息，除非那是一个重要的报告。

RAID 总结见表 1-1。

表 1-1　RIAD 总结

RAID Level	性 能 提 升	冗 余 能 力	利用空间率	磁盘数量（块）
RAID 0	读、写提升	无	100%	至少 2
RAID 1	读性能提升，写性能下降	有	50%	至少 2
RAID 5	读、写提升	有	(n-1)/n%	至少 3
RAID 1+0	读、写提升	有	50%	至少 4
RAID 0+1	读、写提升	有	50%	至少 4
RAID 5+0	读、写提升	有	(n-2)/n%	至少 6

3. RAID 与分区、文件系统的关系

RAID 层是在分区之上的，可以通过对几个分区（通常是几块盘上的）创建 RAID 磁盘阵列，来提高数据的 I/O 速率。对于创建的 RAID，又可以看作是一整块硬盘设备，对该设备进行分区、格式化，挂载使用 RAID 与文件系统。RAID 与分区、文件系统等的关系如图 1-28 所示。

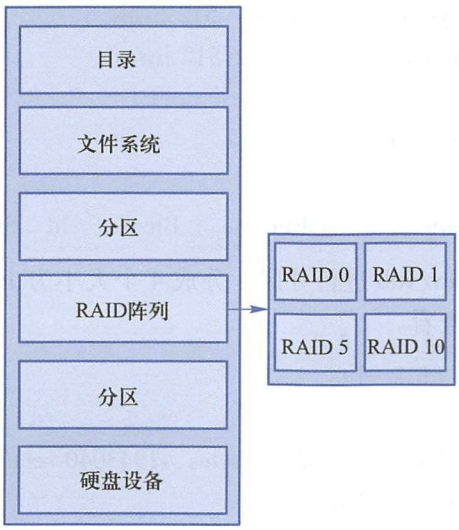


图 1-28　RAID 与分区、文件系统的关系

笔记

任务实施

（1）实验环境

系统：CentOS 7，使用刚才创建的虚拟机来进行操作。

工具：mdadm-3.4-14.el7_3.1.x86_64。

利用磁盘分区新建4个磁盘分区，每个大小为10GB。用这4个10GB的分区来模拟4个1 TB的硬盘。

（2）Fdisk分区

使用Fdisk分区。这里的设备名称为/dev/vdb。

首先用“fdisk -l”命令查看硬盘的情况。

```
[root@localhost ~]# fdisk -l

Disk /dev/vda:21.5 GB,21474836480 bytes,41943040 sectors
Units=sectors of 1*512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0x000af71d

Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *        2048    41929649    20963801   83  Linux

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1*512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

Device Boot      Start         End      Blocks   Id  System
```

使用Fdisk命令对/dev/vdb进行分区，分成4个大小为10 GB的分区，分区完后再使用“fdisk -l”命令查看。

```
[root@localhost ~]# fdisk -l

Disk /dev/vda:21.5 GB,21474836480 bytes,41943040 sectors
Units=sectors of 1*512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
```

```
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0x000af71d

Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *     2048    41929649    20963801   83  Linux

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1 * 512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

Device Boot      Start         End      Blocks   Id  System
/dev/vdb1        2048    20973567    10485760   83  Linux
/dev/vdb2    20973568    41945087    10485760   83  Linux
/dev/vdb3    41945088    62916607    10485760   83  Linux
/dev/vdb4    62916608    83888127    10485760   83  Linux
```

可以使用“lsblk”命令来查看硬盘分区信息。

```
[root@localhost ~]# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda         253:0    0   20G  0 disk
└─vda1      253:1    0   20G  0 part /
vdb         253:16   0   50G  0 disk
├─vdb1      253:17   0   10G  0 part
├─vdb2      253:18   0   10G  0 part
├─vdb3      253:19   0   10G  0 part
└─vdb4      253:20   0   10G  0 part
```

（3）RAID 创建

① 创建一个 RAID 0 设备：这里使用/dev/vdb 和/dev/vdb2 做实验。将/dev/vdb1 和/dev/vdb2 建立 RAID 等级为 0 的 md0（设备名）。

```
[root@localhost ~]# mdadm -Cv /dev/md0 -l0 -n2 /dev/vdb[1-2]
mdadm:chunk size defaults to 512K
mdadm:/dev/vdb1 appears to contain an ext2fs file system
```

```
        size=10485760K  mtime=Thu May 11 02:30:15 2017
Continue creating array? yes
mdadm:Defaulting to version 1.2 metadata
mdadm:array /dev/md0 started.
```

命令解析：

```
-Cv:创建设备,并显示信息。
-l0:RAID 的等级为 RAID 0。
-n2:创建 RAID 的设备为 2 块。
```

查看系统中的 RAID。

```
[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0]
md0:active RAID0 vdb2[1] vdb1[0]
      20955136 blocks super 1.2 512k chunks

unused devices:<none>
```

创建完之后进行删除操作。

```
[root@localhost ~]# mdadm --stop /dev/md0
mdadm:stopped /dev/md0
[root@localhost ~]# mdadm --misc --zero-superblock /dev/vdb1    #移除 vdb1
[root@localhost ~]# mdadm --misc --zero-superblock /dev/vdb2    #移除 vdb2
```

再查看系统中 RAID 情况。

```
[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0]
unused devices:<none>
```

② 创建一个 RAID 1 设备：这里采取/dev/vdb 和/dev/vdb2 两个分区做实验。

```
[root@localhost ~]# mdadm -Cv /dev/md0 -l1 -n2 /dev/vdb[1-2]
mdadm:/dev/vdb1 appears to contain an ext2fs file system
      size=10485760K  mtime=Thu May 11 02:30:15 2017
mdadm:Note:this array has metadata at the start and
may not be suitable as a boot device.  If you plan to
    store '/boot'on this device please ensure that
    your boot-loader understands md/v1.x metadata,or use
    --metadata=0.90
mdadm:size set to 10477568K
```

```
Continue creating array? yes
mdadm:Defaulting to version 1.2 metadata
mdadm:array /dev/md0 started.
```

查看组件 RAID 的进度。

```
[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1]
md0:active RAID1 vdb2[1] vdb1[0]
      10477568 blocks super 1.2 [2/2] [UU]
      [=>.....................]  resync =  9.8% (1028224/10477568) finish=
3.0min speed=51411K/sec

unused devices:<none>

[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1]
md0:active RAID1 vdb2[1] vdb1[0]
      10477568 blocks super 1.2 [2/2] [UU]
      [========>.............]  resync = 35.7% (3742400/10477568)
finish=2.0min speed=55774K/sec

unused devices:<none>

[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1]
md0:active RAID1 vdb2[1] vdb1[0]
      10477568 blocks super 1.2 [2/2] [UU]

unused devices:<none>
```

使用“fdisk -l”命令查看硬盘信息。

```
[root@localhost ~]# fdisk -l

Disk /dev/vda:21.5 GB,21474836480 bytes,41943040 sectors
Units=sectors of 1 * 512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
```

```
Disk identifier:0x000af71d

   Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *        2048    41929649    20963801   83  Linux

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1 * 512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

Device Boot      Start         End      Blocks   Id  System
/dev/vdb1        2048      20973567    10485760   83  Linux
/dev/vdb2        20973568  41945087    10485760   83  Linux
/dev/vdb3        41945088  62916607    10485760   83  Linux
/dev/vdb4        62916608  83888127    10485760   83  Linux

Disk /dev/md0:10.7 GB,10729029632 bytes,20955136 sectors
Units=sectors of 1 * 512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
```

可以看到一块名为/dev/md0 的设备，大小为 10 GB。

③ 创建一个 RAID 5 设备：这里采取/dev/vdb1、/dev/vdb2 和/dev/vdb3 这 3 个分区做实验，按照刚才的方法，先停掉/dev/md0 设备，再移除/dev/vdb1 和/dev/vdb2 这两个分区。

```
[root@localhost ~]# mdadm --stop /dev/md0
mdadm:stopped /dev/md0
[root@localhost ~]# mdadm --misc --zero-superblock /dev/vdb1
[root@localhost ~]# mdadm --misc --zero-superblock /dev/vdb2
[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1]
unused devices:<none>
```

从结果中可以看到创建的 RAID 5 设备不存在了。

现在使用 3 个分区来创建 1 个 RAID 5 设备。

```
[root@localhost ~]# mdadm -Cv /dev/md0 -l5 -n3 /dev/vdb[1-3]
```

```
mdadm:layout defaults to left-symmetric
mdadm:layout defaults to left-symmetric
mdadm:chunk size defaults to 512K
mdadm:/dev/vdb1 appears to contain an ext2fs file system
       size=10485760K  mtime=Thu May 11 02:30:15 2017
mdadm:size set to 10477568K
Continue creating array? yes
mdadm:Defaulting to version 1.2 metadata
mdadm:array /dev/md0 started.
```

查看 RAID 5 构建的进度。

```
[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1] [RAID6] [RAID5] [RAID4]
md0:active RAID5 vdb3[3] vdb2[1] vdb1[0]
      20955136 blocks super 1.2 level 5,512k chunk,algorithm 2 [3/2] [UU_]
      [=>.................]  recovery =  5.4% (571128/10477568) finish
=6.0min speed=27196K/sec

unused devices:<none>

[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1] [RAID6] [RAID5] [RAID4]
md0:active RAID5 vdb3[3] vdb2[1] vdb1[0]
      20955136 blocks super 1.2 level 5,512k chunk,algorithm 2 [3/2] [UU_]
      [==================>...]  recovery = 85.1% (8923932/
10477568) finish=0.9min speed=27930K/sec

unused devices:<none>

[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1] [RAID6] [RAID5] [RAID4]
md0:active RAID5 vdb3[3] vdb2[1] vdb1[0]
      20955136 blocks super 1.2 level 5,512k chunk,algorithm 2 [3/3] [UUU]

unused devices:<none>
```

RAID 已经组建完毕。

使用“fdisk -l”命令查看系统的分区。

```
[root@localhost ~]# fdisk -l

Disk /dev/vda:21.5 GB,21474836480 bytes,41943040 sectors
Units=sectors of 1 * 512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0x000af71d

Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *     2048     41929649    20963801   83  Linux

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1 * 512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

Device Boot      Start         End      Blocks   Id  System
/dev/vdb1        2048     20973567    10485760   83  Linux
/dev/vdb2    20973568     41945087    10485760   83  Linux
/dev/vdb3    41945088     62916607    10485760   83  Linux
/dev/vdb4    62916608     83888127    10485760   83  Linux

Disk /dev/md0:21.5 GB,21458059264 bytes,41910272 sectors
Units=sectors of 1 * 512=512 bytes
Sector size(logical/physical):512 bytes / 512 bytes
I/O size(minimum/optimal):524288 bytes / 1048576 bytes
```

可以看见一块/dev/md0 的设备，使用“mdadm -D”命令查看 RAID 的详细信息。

```
[root@localhost ~]# mdadm -D /dev/md0
/dev/md0:
        Version:1.2
  Creation Time:Thu May 11 03:02:46 2017
     RAID Level:RAID5
```

```
        Array Size : 20955136 (19.98 GiB 21.46 GB)
     Used Dev Size : 10477568 (9.99 GiB 10.73 GB)
      RAID Devices : 3
     Total Devices : 3
       Persistence : Superblock is persistent

       Update Time : Thu May 11 03:09:15 2017
             State : clean
Active Devices : 3
Working Devices : 3
Failed Devices : 0
  Spare Devices : 0

            Layout : left-symmetric
        Chunk Size : 512K

Name : localhost:0  (local to host localhost)
              UUID : f9bc6331:80a54f66:fccd5ab8:9e190c13
            Events : 18

    Number   Major   Minor   RAIDDevice State
       0     253       17        0      active sync   /dev/vdb1
       1     253       18        1      active sync   /dev/vdb2
       3     253       19        2      active sync   /dev/vdb3
```

使用 Ext 4 分区格式化磁盘。

```
[root@localhost ~]# mkfs.ext4 /dev/md0
mke2fs 1.42.9 (28-Dec-2013)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=128 blocks, Stripe width=256 blocks
1310720 inodes, 5238784 blocks
261939 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=2153775104
```

```
160 block groups
32768 blocks per group,32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
  32768,98304,163840,229376,294912,819200,884736,1605632,2654208,
  4096000

Allocating group tables:done
Writing inode tables:done
Creating journal(32768 blocks):done
Writing superblocks and filesystem accounting information:done
```

挂载并验证。

```
[root@localhost ~]# mount /dev/md0 /mnt/
[root@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1       20G   977M  20G   5%  /
devtmpfs        984M  0     984M  0%  /dev
tmpfs           1001M 0     1001M 0%  /dev/shm
tmpfs           1001M 17M   985M  2%  /run
tmpfs           1001M 0     1001M 0%  /sys/fs/cgroup
/dev/md0        20G   45M   19G   1%  /mnt
```

④ 创建 1 个 RAID 10 设备：这里采取/dev/vdb[1-4] 4 个分区做实验。

停止已经创建的 RAID，并移除设备。

```
[root@localhost ~]# umount /mnt/
[root@localhost ~]# mdadm --stop /dev/md0
mdadm:stopped /dev/md0
[root@localhost ~]# mdadm --misc --zero-superblock /dev/vdb1
[root@localhost ~]# mdadm --misc --zero-superblock /dev/vdb2
[root@localhost ~]# mdadm --misc --zero-superblock /dev/vdb3
[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1] [RAID6] [RAID5] [RAID4]
unused devices:<none>
```

创建 RAID 10。

```
[root@localhost ~]# mdadm -Cv /dev/md0 -l10 -n4 /dev/vdb[1-4]
mdadm:layout defaults to n2
```

```
mdadm:layout defaults to n2
mdadm:chunk size defaults to 512K
mdadm:/dev/vdb1 appears to contain an ext2fs file system
       size=10485760K   mtime=Thu May 11 02:30:15 2017
mdadm:size set to 10477568K
Continue creating array? yes
mdadm:Defaulting to version 1.2 metadata
mdadm:array /dev/md0 started.
```

查看 RAID 构建进度。

```
[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1] [RAID6] [RAID5] [RAID4] [RAID10]
md0:active RAID10 vdb4[3] vdb3[2] vdb2[1] vdb1[0]
      20955136 blocks super 1.2 512K chunks 2 near-copies [4/4] [UUUU]
      [======>.............]   resync=30.9%(6477056/20955136)finish
=1.7min speed=136690K/sec

unused devices:<none>

[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1] [RAID6] [RAID5] [RAID4] [RAID10]
md0:active RAID10 vdb4[3] vdb3[2] vdb2[1] vdb1[0]
      20955136 blocks super 1.2 512K chunks 2 near-copies [4/4] [UUUU]

unused devices:<none>
```

这样 RAID 10 就创建好了。关于 RAID 10 的使用，和前面的 RAID 5 大同小异。接下来学习 RAID 的运维。

⑤ RAID 5 的运维操作。首先创建一个 RAID 5，加一个热备盘。

```
[root@localhost ~]# mdadm -Cv /dev/md0 -l5 -n3 /dev/vdb[1-3] --spare-
devices=1 /dev/vdb4
mdadm:layout defaults to left-symmetric
mdadm:layout defaults to left-symmetric
mdadm:chunk size defaults to 512K
mdadm:/dev/vdb1 appears to contain an ext2fs file system
       size=10485760K   mtime=Thu May 11 02:30:15 2017
mdadm:size set to 10477568K
Continue creating array? yes
```

```
mdadm:Defaulting to version 1.2 metadata
mdadm:array /dev/md0 started.
```

查看构建进度。

```
[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1] [RAID6] [RAID5] [RAID4] [RAID10]
md0:active RAID5 vdb3[4] vdb4[3](S)vdb2[1] vdb1[0]
      20955136 blocks super 1.2 level 5,512k chunk,algorithm 2 [3/2] [UU_]
      [>.....................] recovery = 1.7%(187096/10477568)finish=6.4min
speed=26728K/sec

unused devices:<none>

[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1] [RAID6] [RAID5] [RAID4] [RAID10]
md0:active RAID5 vdb3[4] vdb4[3](S)vdb2[1] vdb1[0]
      20955136 blocks super 1.2 level 5,512k chunk,algorithm 2 [3/3] [UUU]

unused devices:<none>
```

构建完成之后，查看 RAID 的详细信息。

```
[root@localhost ~]# mdadm -D /dev/md0
/dev/md0:
        Version:1.2
  Creation Time:Thu May 11 03:31:06 2017
     RAID Level:RAID5
     Array Size:20955136(19.98 GiB 21.46 GB)
  Used Dev Size:10477568(9.99 GiB 10.73 GB)
   RAID Devices:3
  Total Devices:4
    Persistence:Superblock is persistent

    Update Time:Thu May 11 03:37:56 2017
          State:clean
 Active Devices:3
 Working Devices:4
 Failed Devices:0
  Spare Devices:1
```

```
          Layout : left-symmetric
       Chunk Size : 512K

Name : localhost : 0  (local to host localhost)
              UUID : 7f45a44b : c641bbcd : 58f6466b : 47a1e4e8
            Events : 18

    Number   Major   Minor   RAIDDevice    State
       0       253     17          0      active sync   /dev/vdb1
       1       253     18          1      active sync   /dev/vdb2
       4       253     19          2      active sync   /dev/vdb3

       3       253     20          -      spare         /dev/vdb4
```

可以看到除了 3 个 active 的设备之外，还有一个备用盘/dev/vdb4。

下一步模拟硬盘故障。

```
[root@localhost ~]# mdadm -f /dev/md0 /dev/vdb1
mdadm : set /dev/vdb1 faulty in /dev/md0
```

再查看 RAID 的构建信息。

```
[root@localhost ~]# cat /proc/mdstat
Personalities : [RAID0] [RAID1] [RAID6] [RAID5] [RAID4] [RAID10]
md0 : active RAID5 vdb3[4] vdb4[3] vdb2[1] vdb1[0](F)
      20955136 blocks super 1.2 level 5,512k chunk,algorithm 2 [3/2] [_UU]
      [>....................] recovery = 1.2%(128740/10477568)finish=6.6min
speed=25748K/sec

unused devices : <none>
```

发现正在重建 RAID，再查看 RAID 的详细信息。

```
[root@localhost ~]# mdadm -D /dev/md0
/dev/md0 :
           Version : 1.2
     Creation Time : Thu May 11 03 : 31 : 06 2017
        RAID Level : RAID5
        Array Size : 20955136 (19.98 GiB 21.46 GB)
     Used Dev Size : 10477568 (9.99 GiB 10.73 GB)
```

```
     RAID Devices:3
   Total Devices:4
         Persistence:Superblock is persistent

         Update Time:Thu May 11 05:59:09 2017
               State:clean,degraded,recovering
    Active Devices:2
   Working Devices:3
    Failed Devices:1
     Spare Devices:1

            Layout:left-symmetric
        Chunk Size:512K

    Rebuild Status:59% complete

Name:localhost:0  (local to host localhost)
              UUID:7f45a44b:c641bbcd:58f6466b:47a1e4e8
            Events:29

    Number   Major   Minor   RAIDDevice   State
       3       253      20        0      spare rebuilding   /dev/vdb4 #重建 RAID5
       1       253      18        1      active sync        /dev/vdb2
       4       253      19        2      active sync        /dev/vdb3

       0       253      17        -      faulty             /dev/vdb1
```

从以上结果可以看出，原来的热备盘/dev/vdb4 正在参与 RAID 5 的重建，而原来的/dev/vdb1 变成了坏盘。再查看以下 RAID 构建的信息。

```
[root@localhost ~]# cat /proc/mdstat
Personalities:[RAID0] [RAID1] [RAID6] [RAID5] [RAID4] [RAID10]
md0:active RAID5 vdb3[4] vdb4[3] vdb2[1] vdb1[0](F)
      20955136 blocks super 1.2 level 5,512k chunk,algorithm 2 [3/3] [UUU]

unused devices:<none>
```

这时，RAID 已经构建完毕，完成后再次查看 RAID 的详细信息。

```
[root@localhost ~]# mdadm -D /dev/md0
/dev/md0:
```

```
        Version : 1.2
  Creation Time : Thu May 11 03:31:06 2017
     RAID Level : RAID5
     Array Size : 20955136 (19.98 GiB 21.46 GB)
  Used Dev Size : 10477568 (9.99 GiB 10.73 GB)
   RAID Devices : 3
  Total Devices : 4
    Persistence : Superblock is persistent

    Update Time : Thu May 11 06:02:03 2017
          State : clean
 Active Devices : 3
Working Devices : 3
 Failed Devices : 1
  Spare Devices : 0

         Layout : left-symmetric
     Chunk Size : 512K

Name : localhost:0  (local to host localhost)
           UUID : 7f45a44b:c641bbcd:58f6466b:47a1e4e8
         Events : 37

    Number   Major   Minor   RaidDevice State
       3       253      20        0      active sync   /dev/vdb4 #同步完成
       1       253      18        1      active sync   /dev/vdb2
       4       253      19        2      active sync   /dev/vdb3

       0       253      17        -      faulty        /dev/vdb1 #故障盘
```

热移除故障盘。

```
[root@localhost ~]# mdadm -r /dev/md0 /dev/vdb1
mdadm: hot removed /dev/vdb1 from /dev/md0
```

查看 RAID 的详细信息。

```
[root@localhost ~]# mdadm -D /dev/md0
/dev/md0:
        Version : 1.2
  Creation Time : Thu May 11 03:31:06 2017
```

```
        RAID Level:RAID5
        Array Size:20955136 (19.98 GiB 21.46 GB)
     Used Dev Size:10477568 (9.99 GiB 10.73 GB)
        RAID Devices:3
     Total Devices:3
       Persistence:Superblock is persistent

       Update Time:Thu May 11 06:11:00 2017
             State:clean
    Active Devices:3
   Working Devices:3
    Failed Devices:0
     Spare Devices:0

            Layout:left-symmetric
        Chunk Size:512K

Name:localhost:0  (local to host localhost)
              UUID:7f45a44b:c641bbcd:58f6466b:47a1e4e8
            Events:38

    Number   Major   Minor   RAIDDevice State
       3       253     20        0      active sync   /dev/vdb4
       1       253     18        1      active sync   /dev/vdb2
       4       253     19        2      active sync   /dev/vdb3
```

格式化 RAID 并进行挂载。

```
[root@localhost ~]# mkfs.ext4 /dev/md0
mke2fs 1.42.9 (28-Dec-2013)
Filesystem label=
OS type:Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=128 blocks,Stripe width=256 blocks
1310720 inodes,5238784 blocks
261939 blocks (5.00%) reserved for the super user
First data block=0
```

```
Maximum filesystem blocks=2153775104
160 block groups
32768 blocks per group,32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
    32768,98304,163840,229376,294912,819200,884736,1605632,2654208,
    4096000

Allocating group tables:done
Writing inode tables:done
Creating journal (32768 blocks):done
Writing superblocks and filesystem accounting information:done

[root@localhost ~]# mount /dev/md0 /mnt/
[root@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        20G  977M   20G   5% /
devtmpfs        984M     0  984M   0% /dev
tmpfs          1001M     0 1001M   0% /dev/shm
tmpfs          1001M   17M  985M   2% /run
tmpfs          1001M     0 1001M   0% /sys/fs/cgroup
/dev/md0         20G   45M   19G   1% /mnt
```

设备可以正常使用。

项目实训

【实训题目】

使用 3 个分区创建 1 个 RAID 5 并挂载使用。模拟坏盘并替换。

【实训目的】

1. 掌握 RAID 的构建、挂载和使用。
2. 掌握 RAID 的基础运维。

【实训内容】

1. 使用 Fdisk 分区工具对 1 个硬盘进行分区，分出 3 个 10 GB 的分区。
2. 使用这 3 个分区进行构建 RAID 5，并进行格式化、挂载。
3. 模拟坏盘，并使用一个新的分区替换原来的坏盘。

任务 1.3 管理逻辑卷

任务描述

1. 了解 LVM 卷以及卷的操作和管理方法。
2. 掌握逻辑卷的构建、操作和使用方法。

知识学习

1. LVM 卷的定义

笔记

LVM（Logical Volume Manager）是 Linux 环境下对磁盘分区进行管理的一种机制，是建立在硬盘和分区之上的一个逻辑层，用来提高磁盘分区管理的灵活性。相关逻辑设备如下。

① 物理卷（Pysical Volume，PV）：也就是物理磁盘，类似于/dev/sda2、/dev/sdb1 等，由物理磁盘块（Physical Extends，PE）组成。多个 PV 可以组合起来形成一个卷组。

② 卷组（Voume Group，VG）：不能直接使用，要想使用组合后的空间需要创建逻辑卷。VG 大小取决与 PV，VG 中可以划分多个逻辑卷（可以动态缩展）。

③ 逻辑卷（Logical Volume）：可以是卷组所有可用空间，本身有物理边界和逻辑边界两种边界；也可以说每个逻辑卷就是个文件系统，或者是个独立的分区。对卷创建了快照实际上是对逻辑卷创建快照，快照卷也就是跟它的逻辑卷在同一个卷组当中。

④ 快照：可以理解成访问同一个文件的另一条途径，把数据停留在过去的某一个时刻主要是实现数据备份。

2. 相关命令

（1）物理卷命令

其命令有 pvcreate（创建 PV）、pvs（查看 PV 信息）、pvdisplay（查看 PV 详细信息）、pvmove（将 PV 数据转移至其他 PV）、pvremove（删除 PV）和 pvscan（扫描 PV）等。

pvcreate：创建 PV 物理卷，如 pvcreate /dev/sdb1。

pvs：查看 PV 物理卷，如 pvs /dev/sdb1。

pvdisplay：查看 PV 详细信息，如 pvdisplay /dev/sdb1。

pvmove：将 PV 物理卷上的数据移动到其他 PV，如 pvmove /path/to/pv。

（2）卷组命令

其命令有 vgcreate（创建 VG）、vgremove（删除 VG）、vgextend（扩展 VG）、vgreduce（缩减 VG）、vgs（查看 VG 信息）、vgdisplay（查看 VG 详细信息）和 vgscan（扫描 VG）等。

-s：指定 PE。

vgcreate myvg /dev/sdb{n,n}：创建 VG 卷组。

vgdisplay myvg：只查看 myvg 卷组。

vgremove myvg：删除 myvg 卷组（删除一个 VG）。

vgreduce VG_NAME /path/to/pv：缩小减 VG，也就意味着可以把 PV 缩减（一定要先把 PV 上的数据移走）。

pvmove /dev/sdb1：把 sdb1 上面的数据移走。

vgreduce myvg /dev/sdb1：从 myvg 移走/dev/sdb1。

pvremove /dev/sdb1：删除/dev/sdb1。

vgextend myvg /dev/sdb2：扩展 myvg 卷组（扩展 VG）。

（3）逻辑卷命令

其命令有 lvcreate（创建 LV）、lvremove（删除 LV）、lvextend（扩展 LV）、lvreduce（缩减 LV）、lvs（查看 LV 信息）、lvdisplay（查看 LV 详细信息）和 lvscan（扫描 LV）等。

lvcreate -L+G：指定空间大小。

-n LV_NAME：显示逻辑卷名称。

lvdisplay：显示所有 LV 逻辑卷，如 lvdisplay dev/myvg/testlv。

笔记

注：这里命令的使用不再一一列举，使用方法都相似。

（4）逻辑卷扩展与缩减要求

① 扩展要求：扩展之前先检查文件系统；逻辑卷边界是紧靠物理卷边界上创建的；先扩展物理卷边界，再扩展逻辑卷边界。

② 缩减逻辑卷要求：不能在线缩减，需要先卸载；确保缩减后的空间大小依然能存储原有的数据；在缩减之前应该强行检查文件，以确保文件系统能正常使用。

3. LVM 卷和分区、文件系统的关系

LVM 卷是建立在磁盘分区上对分区的一种管理，首先把磁盘分区创建为物理卷，然后将物理卷组成卷组，最后在卷组中创建出 LVM 逻辑卷。在 LVM 卷的基础上又可以进行格式化文件系统，然后挂载使用。LVM 逻辑卷与分区、文件系统等的结构关系如图 1-29 所示。

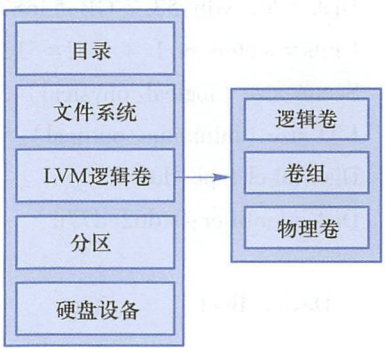


图 1-29　LVM 卷结构关系

任务实施

① 搭建实验环境。

系统：CentOS 7。

创建一个 10 GB 的 VG。

② 创建分区。在/dev/vdb 硬盘上创建两个主分区，分别为 5 GB。

首先，删除/dev/vdb 上所有的分区。

```
[root@localhost ~]# fdisk /dev/vdb
Welcome to fdisk (util-linux 2.23.2).
Changes will remain in memory only,until you decide to write them.
Be careful before using the write command.
Command (m for help):d
Partition number (1-4,default 4):
Partition 4 is deleted

Command (m for help):d
Partition number (1-3,default 3):
Partition 3 is deleted

Command (m for help):d
Partition number (1,2,default 2):
Partition 2 is deleted

Command (m for help):d
Selected partition 1
Partition 1 is deleted

Command (m for help):p

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1 * 512=512 bytes
Sector size (logical/physical):512 bytes / 512 bytes
I/O size (minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

   Device Boot      Start         End      Blocks   Id  System

Command (m for help):w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

在删除时，选择删除哪个分区即可。这里是将创建的 4 个分区全部删除，选中分区直接按〈Enter〉键即可。删除完分区后，使用“lsblk”命令查看。

```
[root@localhost ~]# lsblk
NAME    MAJ:MIN  RM  SIZE  RO  TYPE  MOUNTPOINT
vda     253:0    0   20G   0   disk
└─vda1  253:1    0   20G   0   part  /
vdb     253:16   0   50G   0   disk
```

③ 分区删除后，格式化 vdb 磁盘。

```
[root@localhost ~]# fdisk /dev/vdb
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only,until you decide to write them.
Be careful before using the write command.

Command (m for help):p

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1 * 512=512 bytes
Sector size (logical/physical):512 bytes / 512 bytes
I/O size (minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

   Device Boot      Start         End      Blocks   Id  System

Command (m for help):n
Partition type:
   p   primary (0 primary,0 extended,4 free)
   e   extended
Select (default p):
Using default response p
Partition number (1-4,default 1):
First sector (2048-104857599,default 2048):
Using default value 2048
Last sector,+sectors or+size{K,M,G} (2048-104857599,default 104857599):+5G
Partition 1 of type Linux and of size 5 GiB is set

Command (m for help):n
```

```
Partition type:
   p   primary (1 primary,0 extended,3 free)
   e   extended
Select (default p):p
Partition number (2-4,default 2):
First sector (10487808-104857599,default 10487808):
Using default value 10487808
Last sector,+sectors or +size{K,M,G} (10487808-104857599,default 104857599):
+5G
Partition 2 of type Linux and of size 5 GiB is set

Command (m for help):p

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1 * 512=512 bytes
Sector size (logical/physical):512 bytes / 512 bytes
I/O size (minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

Device Boot   Start      End        Blocks    Id  System
/dev/vdb1     2048       10487807   5242880   83  Linux
/dev/vdb2     10487808   20973567   5242880   83  Linux

Command (m for help):w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

笔记

创建完成后用“lsblk”命令查看。

```
[root@localhost ~]# lsblk
NAME    MAJ:MIN  RM  SIZE  RO  TYPE  MOUNTPOINT
vda     253:0    0   20G   0   disk
└─vda1  253:1    0   20G   0   part  /
vdb     253:16   0   50G   0   disk
├─vdb1  253:17   0   5G    0   part
└─vdb2  253:18   0   5G    0   part
```

创建的两个主分区为/dev/vdb1 和/dev/vdb2 ，大小都是 5 GB。

④ 让内核重新读取分区并查看。

```
[root@localhost ~]# partx /dev/vdb
NR  START     END       SECTORS   SIZE NAME UUID
1   2048      10487807  10485760  5G
2   10487808  20973567  10485760  5G
```

⑤ 创建物理卷组。

首先，创建 PV 物理卷。

```
[root@localhost ~]# pvcreate /dev/vdb1 /dev/vdb2
  Physical volume "/dev/vdb1" successfully created.
  Physical volume "/dev/vdb2" successfully created.
```

查看创建的物理卷。

```
[root@localhost ~]# pvs
  PV         VG Fmt  Attr PSize PFree
  /dev/vdb1     lvm2 ---  5.00g 5.00g
  /dev/vdb2     lvm2 ---  5.00g 5.00g
```

查看物理卷的详细信息。

```
[root@localhost ~]# pvdisplay
"/dev/vdb2" is a new physical volume of "5.00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/vdb2
  VG Name
  PV Size               5.00 GiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               UiHBiw-ZvNB-HgH0-tgyA-gPgL-Ia3v-M2TSqb

"/dev/vdb1" is a new physical volume of "5.00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/vdb1
  VG Name
  PV Size               5.00 GiB
  Allocatable           NO
```

```
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               MvQkw6-0SMY-kqiK-Ufjg-4JV0-pD7T-CCedyb
```

扫描物理卷。

```
[root@localhost ~]# pvscan
  PV /dev/vdb2                        lvm2 [5.00 GiB]
  PV /dev/vdb1                        lvm2 [5.00 GiB]
  Total:2 [10.00 GiB] / in use:0 [0   ] / in no VG:2 [10.00 GiB]
```

创建名称为 myvg 的卷组。

```
[root@localhost ~]# vgcreate myvg /dev/vdb[1-2]
  Volume group "myvg" successfully created
```

查看卷组信息。

```
[root@localhost ~]# vgs
    VG   #PV #LV #SN Attr   VSize VFree
    myvg   2   0   0 wz--n- 9.99g 9.99g
```

查看卷组的详细信息。

```
[root@localhost ~]# vgdisplay
  --- Volume group ---
  VG Name               myvg
  System ID
  Format                lvm2
  Metadata Areas        2
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                2
  Act PV                2
  VG Size               9.99 GiB
  PE Size               4.00 MiB
```

```
  Total PE              2558
  Alloc PE / Size       0 / 0
  Free  PE / Size       2558 / 9.99 GiB
  VG UUID               7EOUYo-Wc6f-BsQO-c2bc-LZpR-8D5C-oPlmvq
```

⑥ 卷组删除。可以看到 PE Size（物理区块大小）是 4 MB，现在删除该卷组，并重新创建一个，指定 PE 的大小为 16 MB。

笔记

删除 myvg。

```
[root@localhost ~]# vgremove myvg
  Volume group "myvg" successfully removed
```

创建 VG 并指定 PE 大小。

```
[root@localhost ~]# vgcreate -s 16m myvg /dev/vdb[1-2]
  Volume group "myvg" successfully created
```

查看卷组详细信息。

```
[root@localhost ~]# vgdisplay
  --- Volume group ---
  VG Name               myvg
  System ID
  Format                lvm2
  Metadata Areas        2
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                2
  Act PV                2
  VG Size               9.97 GiB
  PE Size               16.00 MiB
  Total PE              638
  Alloc PE / Size       0 / 0
  Free  PE / Size       638 / 9.97 GiB
  VG UUID               DbZgNW-SIAK-Q5cd-gsRX-Kwtj-6hkI-cFEK7Y
```

可以看到 PE Size 为 16 MB。

从卷组中删除 PV，先转移数据。

```
[root@localhost ~]# pvmove /dev/vdb1
  No data to move for myvg
```

从 myvg 中删除/dev/vdb1。

```
[root@localhost ~]# vgreduce myvg /dev/vdb1
  Removed "/dev/vdb1" from volume group "myvg"
```

查看物理卷信息。

```
[root@localhost ~]# pvs
  PV         VG   Fmt  Attr PSize PFree
  /dev/vdb1       lvm2 ---  5.00g 5.00g
  /dev/vdb2  myvg lvm2 a--  4.98g 4.98g
```

⑦ 添加物理卷。向卷组 myvg 中添加一个物理卷，在/dev/vdb 上再分一个/dev/vdb3 分区，把该分区加到卷组 myvg 中。

```
[root@localhost ~]# vgextend myvg /dev/vdb3
  Physical volume "/dev/vdb3" successfully created.
  Volume group "myvg" successfully extended
```

查看物理卷信息。

```
[root@localhost ~]# pvs
  PV         VG   Fmt  Attr PSize PFree
  /dev/vdb1       lvm2 ---  5.00g 5.00g
  /dev/vdb2  myvg lvm2 a--  4.98g 4.98g
  /dev/vdb3  myvg lvm2 a--  4.98g 4.98g
```

至此卷组已经创建完毕，接下来需要完成逻辑卷的创建。

⑧ 创建逻辑卷。名为 mylv，大小为 5 GB。

```
[root@localhost ~]# lvcreate -L+5G -n mylv myvg
  Logical volume "mylv" created.
```

-L：创建逻辑卷的大小 large。

-n：创建的逻辑卷名称。

查看逻辑卷。

```
[root@localhost ~]# lvs
  LV   VG   Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%
Sync Convert
  mylv myvg -wi-a----- 5.00g
```

查看逻辑卷详细信息。

```
[root@localhost ~]# lvdisplay
  --- Logical volume ---
  LV Path                /dev/myvg/mylv
  LV Name                mylv
  VG Name                myvg
  LV UUID                lcaYkG-SYfb-Ccvh-hXV6-xpF3-z7L1-4C9sf7
  LV Write Access        read/write
  LV Creation host,time localhost,2017-05-11 07:07:32+0000
  LV Status              available
  # open                 0
  LV Size                5.00 GiB
  Current LE             320
  Segments               2
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     8192
  Block device           252:0
```

扫描上一步创建的逻辑卷。

```
[root@localhost ~]# lvscan
  ACTIVE            '/dev/myvg/mylv' [5.00 GiB] inherit
```

这里可以使用“fdisk -l”命令来查看创建的卷设备。

```
[root@localhost ~]# fdisk -l

Disk /dev/vda:21.5 GB,21474836480 bytes,41943040 sectors
Units=sectors of 1 * 512=512 bytes
Sector size (logical/physical):512 bytes / 512 bytes
I/O size (minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0x000af71d

Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *     2048    41929649    20963801   83  Linux

Disk /dev/vdb:53.7 GB,53687091200 bytes,104857600 sectors
Units=sectors of 1 * 512=512 bytes
```

```
Sector size (logical/physical):512 bytes / 512 bytes
I/O size (minimum/optimal):512 bytes / 512 bytes
Disk label type:dos
Disk identifier:0xd62cd77d

Device Boot      Start         End      Blocks   Id  System
/dev/vdb1        2048      10487807     5242880   8e  Linux LVM
/dev/vdb2    10487808      20973567     5242880   8e  Linux LVM
/dev/vdb3    20973568      31459327     5242880   8e  Linux LVM

Disk /dev/mapper/myvg-mylv:5368 MB,5368709120 bytes,10485760 sectors
Units=sectors of 1 * 512=512 bytes
Sector size (logical/physical):512 bytes / 512 bytes
I/O size (minimum/optimal):512 bytes / 512 bytes
```

⑨ 格式化卷组。

首先，格式化逻辑卷 mylv。

笔记

```
[root@localhost ~]# mkfs.ext4 /dev/mapper/myvg-mylv
mke2fs 1.42.9 (28-Dec-2013)
Filesystem label=
OS type:Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks,Stripe width=0 blocks
327680 inodes,1310720 blocks
65536 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1342177280
40 block groups
32768 blocks per group,32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768,98304,163840,229376,294912,819200,884736

Allocating group tables:done
Writing inode tables:done
Creating journal (32768 blocks):done
Writing superblocks and filesystem accounting information:done
```

把逻辑卷 mylv 挂载到/mnt 下并验证。

```
[root@localhost ~]# mount /dev/mapper/myvg-mylv /mnt/
[root@localhost ~]# df -h
Filesystem              Size  Used Avail Use% Mounted on
/dev/vda1               20G  1.1G   19G   6% /
devtmpfs                984M     0  984M   0% /dev
tmpfs                   1001M    0  1001M  0% /dev/shm
tmpfs                   1001M  25M  977M   3% /run
tmpfs                   1001M    0  1001M  0% /sys/fs/cgroup
/dev/mapper/myvg-mylv   4.8G   20M  4.6G   1% /mnt
```

⑩ 卷组扩容。

接下来，对创建的 LVM 卷扩容 1 GB。

```
[root@localhost ~]# lvextend -L+1G /dev/mapper/myvg-mylv
  Size of logical volume myvg/mylv changed from 5.00 GiB (320 extents) to 6.00 GiB (384 extents).
  Logical volume myvg/mylv successfully resized.
[root@localhost ~]# lvs
  LV   VG   Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  mylv myvg -wi-ao---- 6.00g
[root@localhost ~]# df -h
Filesystem              Size  Used Avail Use% Mounted on
/dev/vda1               20G  1.1G   19G   6% /
devtmpfs                984M     0  984M   0% /dev
tmpfs                   1001M    0  1001M  0% /dev/shm
tmpfs                   1001M  25M  977M   3% /run
tmpfs                   1001M    0  1001M  0% /sys/fs/cgroup
/dev/mapper/myvg-mylv   4.8G   20M  4.6G   1% /mnt
```

可以看到 LVM 卷的大小变成了 6 GB，但是挂载信息中没有发生变化，这时系统还识别添加进来的磁盘文件系统，所以还需要对文件系统进行扩容。

```
[root@localhost ~]# resize2fs /dev/mapper/myvg-mylv
resize2fs 1.42.9 (28-Dec-2013)
Filesystem at /dev/mapper/myvg-mylv is mounted on /mnt; on-line resizing required
old_desc_blocks=1, new_desc_blocks=1
The filesystem on /dev/mapper/myvg-mylv is now 1572864 blocks long.
[root@localhost ~]# df -h
```

```
Filesystem              Size  Used  Avail  Use%  Mounted on
/dev/vda1               20G   1.1G  19G    6%    /
devtmpfs                984M  0     984M   0%    /dev
tmpfs                   1001M 0     1001M  0%    /dev/shm
tmpfs                   1001M 25M   977M   3%    /run
tmpfs                   1001M 0     1001M  0%    /sys/fs/cgroup
/dev/mapper/myvg-mylv   5.8G  20M   5.5G   1%    /mnt
```

操作完成后看到实际的容量变成了 6 GB。如果需要创建一个 11 GB 大小的逻辑卷，而当前的卷组才 10 GB，不够创建这么大的逻辑卷，所以首先要对卷组进行扩容。

```
[root@localhost ~]# vgextend myvg /dev/vdb1
  Volume group "myvg" successfully extended
```

查看物理卷信息和卷组信息。

```
[root@localhost ~]# vgs
  VG   #PV #LV #SN Attr   VSize   VFree
  myvg   3   1   0 wz--n- 14.95g 8.95g
[root@localhost ~]# pvs
  PV         VG   Fmt  Attr  PSize  PFree
  /dev/vdb1  myvg lvm2 a--   4.98g  4.98g
  /dev/vdb2  myvg lvm2 a--   4.98g  0
  /dev/vdb3  myvg lvm2 a--   4.98g  3.97g
```

接下来，对逻辑卷进行扩容 5 GB。

```
[root@localhost ~]# lvextend -L+5G /dev/mapper/myvg-mylv
  Size of logical volume myvg/mylv changed from 6.00 GiB (384 extents) to 11.00 GiB (704 extents).
  Logical volume myvg/mylv successfully resized.
[root@localhost ~]# lvs
  LV   VG   Attr   LSize  Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  mylv myvg -wi-ao---- 11.00g
[root@localhost ~]# df -h
Filesystem              Size  Used  Avail  Use%  Mounted on
/dev/vda1               20G   1.1G  19G    6%    /
devtmpfs                984M  0     984M   0%    /dev
tmpfs                   1001M 0     1001M  0%    /dev/shm
tmpfs                   1001M 25M   977M   3%    /run
tmpfs                   1001M 0     1001M  0%    /sys/fs/cgroup
/dev/mapper/myvg-mylv   5.8G  20M   5.5G   1%    /mnt
```

再对文件系统进行扩容。

```
[root@localhost ~]# resize2fs /dev/mapper/myvg-mylv
resize2fs 1.42.9 (28-Dec-2013)
Filesystem at /dev/mapper/myvg-mylv is mounted on /mnt; on-line resizing required
old_desc_blocks=1, new_desc_blocks=2
The filesystem on /dev/mapper/myvg-mylv is now 2883584 blocks long.
[root@localhost ~]# df -h
Filesystem              Size  Used  Avail  Use%  Mounted on
/dev/vda1               20G   1.1G  19G    6%    /
devtmpfs                984M  0     984M   0%    /dev
tmpfs                   1001M 0     1001M  0%    /dev/shm
tmpfs                   1001M 25M   977M   3%    /run
tmpfs                   1001M 0     1001M  0%    /sys/fs/cgroup
/dev/mapper/myvg-mylv   11G   25M   11G    1%    /mnt
```

成功地扩容逻辑卷到 11 GB。

笔记

⑪ 卷组缩减大小。完成了卷的扩容之后，再次尝试将逻辑卷缩减 3 GB 的操作。使用“lvreduce”命令收缩逻辑卷的空间大小有可能会删除逻辑卷上已有的数据，所以在操作前必须进行确认。首先将已经挂载的逻辑卷卸载，需要注意的是“lvreduce”命令不能在线操作。

```
[root@localhost ~]# umount /mnt/
[root@localhost ~]# df -h
Filesystem      Size  Used  Avail  Use%  Mounted on
/dev/vda1       20G   1.1G  19G    6%    /
devtmpfs        984M  0     984M   0%    /dev
tmpfs           1001M 0     1001M  0%    /dev/shm
tmpfs           1001M 25M   977M   3%    /run
tmpfs           1001M 0     1001M  0%    /sys/fs/cgroup
```

缩减逻辑卷的大小。

```
[root@localhost ~]# lvreduce -L -3G /dev/mapper/myvg-mylv
  WARNING: Reducing active logical volume to 8.00 GiB.
  THIS MAY DESTROY YOUR DATA (filesystem etc.)
Do you really want to reduce myvg/mylv? [y/n]: y
  Size of logical volume myvg/mylv changed from 11.00 GiB (704 extents) to 8.00
GiB (512 extents).
  Logical volume myvg/mylv successfully resized.
[root@localhost ~]# lvs
```

```
  LV   VG   Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%
Sync Convert
  mylv myvg -wi-a----- 8.00g
```

可以看到逻辑卷的大小变为了 8 GB。

格式化逻辑卷并挂载逻辑卷。

```
[root@localhost ~]# mkfs.ext4 /dev/mapper/myvg-mylv
mke2fs 1.42.9 (28-Dec-2013)
Filesystem label=
OS type:Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks,Stripe width=0 blocks
524288 inodes,2097152 blocks
104857 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=2147483648
64 block groups
32768 blocks per group,32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
    32768,98304,163840,229376,294912,819200,884736,1605632

Allocating group tables:done
Writing inode tables:done
Creating journal (32768 blocks):done
Writing superblocks and filesystem accounting information:done

[root@localhost ~]# mount /dev/mapper/myvg-mylv /mnt/
[root@localhost ~]# df -h
Filesystem              Size  Used Avail Use% Mounted on
/dev/vda1               20G   1.1G  19G   6%  /
devtmpfs                984M  0     984M  0%  /dev
tmpfs                   1001M 0     1001M 0%  /dev/shm
tmpfs                   1001M 25M   977M  3%  /run
tmpfs                   1001M 0     1001M 0%  /sys/fs/cgroup
/dev/mapper/myvg-mylv   7.8G  36M   7.3G  1%  /mnt
```

缩减逻辑卷操作成功。

项目实训

【实训题目】

使用两个 10 GB 的分区创建一个卷组，并在这个卷组中创建一个大小为 15 GB 的逻辑卷，接着对该逻辑卷挂载使用。最后把逻辑卷大小扩大 1 GB。

【实训目的】

1. 掌握物理卷、卷组、逻辑卷的构建。
2. 掌握逻辑卷的运维，会在线扩大或缩小逻辑卷的大小，并灵活运用。

【实训内容】

1. 使用 Fdisk 分区工具分出两个 10 GB 的分区。
2. 把这两个分区创建为物理卷，并组成卷组。
3. 在卷组中创建一个大小为 15 GB 的逻辑卷，并进行格式化、挂载。
4. 对逻辑卷进行扩容，先扩容空间，再扩容文件系统。

单元小结

本单元主要学习了服务器的内置存储系统，包括对硬盘的分区、格式化、挂载操作。使用分区进行构建 RAID，并对 RAID 进行运维操作。使用分区构建卷组、逻辑卷，并进行运维操作。这些内置存储技术是学习存储知识的基础。学好这些技术，对于以后学习 NFS、CIFS 以及 iSCSI 等存储技术都是很有帮助的。

单元 2
构建外置存储系统

学习目标

【知识目标】

PPT2：构建外置存储系统

- 了解外置存储技术的种类。
- 了解外置存储技术的特性。
- 了解储存虚拟化技术。
- 了解 Openfiler 技术。
- 了解 NAS 网络存储器。
- 了解 iSCSI 的工作过程。

【技能目标】

- 掌握使用 CentOS 操作系统构建 NFS。
- 掌握使用 CentOS 操作系统构建 CIFS。
- 掌握 Openfiler 工具的部署。
- 掌握使用 Openfiler 工具搭建 NAS 存储网络。
- 掌握使用 Openfiler 工具搭建 iSCSI 存储网络。

学习情境

研发部的小缪在内置存储前期的使用中意识到内置存储并不能满足公司的要求。在已有设计的基础上，他建议重新设计公司的数据服务器，并采用数据集中存储方案进行建设。于是，研发部让小缪按照以下要求进行环境规划和准备。

（1）实现存储基本规划

① 使用 CentOS 实现 NFS 服务配置。

② 使用 CentOS 实现 CIFS 服务配置。

③ 使用 CentOS 实现 iSCSI 服务配置。

（2）服务器功能实现

① 实现基于 CentOS 系统的 NAS 和 iSCSI 的共享存储。

② 使用 Openfiler 实现 NAS 和 iSCSI。

任务 2.1 了解外置存储技术

任务描述

1. 了解不同外置存储技术的优缺点。
2. 使用 CentOS 构建 NFS 和 CIFS。
3. 使用 CentOS 构建 iSCSI。

知识学习

笔 记

1. 外置存储基本技术概述

随着主机、磁盘、网络等技术的发展，数据存储的方式和架构也在一直不停地改变。本单元主要介绍目前主流的存储架构。

外挂存储根据连接的方式分为直连式存储（Direct-Attached Storage，DAS）和网络化存储（Fabric-Attached Storage，FAS）。网络化存储根据传输协议又分为网络接入存储（Network-Attached Storage，NAS）和存储区域网络（Storage Area Network，SAN）。

常见的 3 种存储架构 DAS、NAS、SAN 比较如图 2-1 所示。

（1）DAS

DAS 在人们的生活中非常常见，尤其在中小企业的应用中，DAS 是最主要的应用模式，存储系统被直连到应用的服务器中。在中小企业中，许多数据应用必须安装在直连的 DAS 存储器上。

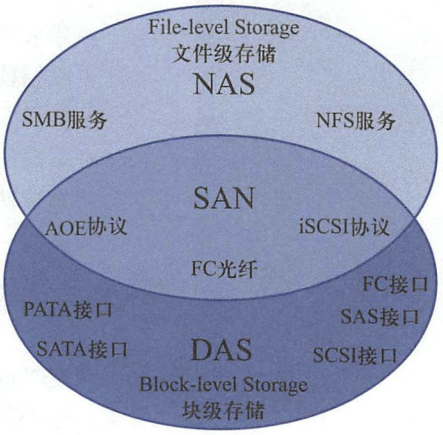


图 2-1 3 种存储架构

DAS 更多依赖服务器主机进行数据的 IO 读写和存储维护管理。数据备份和恢复要求占用服务器主机资源（包括 CPU、系统 IO 等），数据流需要回流主机再到服务器连接着的磁带机（库）。数据备份通常需要占用服务器主机资源的 20%～30%，因此许多企业用户的日常数据备份常常在深夜或业务系统不繁忙时进行，以免影响正常业务。DAS 的数据量越大，备份和恢复的时间就越长，对服务器硬件的依赖性和影响就越大。

DAS 与服务器主机之间的连接通道通常采用 SCSI（Small Computer System Interface，小型计算机系统接口）连接。随着服务器 CPU 的处理能力越来越强，存储硬盘空间越来越大，阵列的硬盘数量越来越多，SCSI 通道将会成为 IO 瓶颈。服务器主机的 SCSI ID 资源有限，能够建立的 SCSI 通道连接也有限。

无论 DAS 还是服务器主机的扩展，从一台服务器扩展为多台服务器组成的群集（Cluster）或存储阵列容量的扩展，都会造成业务系统的停机，从而给企业带来经济损失。例如，银行、电信、传媒等行业“7×24 小时”服务的关键业务系统，若造成服务中断，人们是不能接受的。而且 DAS 或服务器主机的升级扩展，往往受原设备厂商的限制，只能由原设备厂商提供。

（2）NAS

NAS 也通常被称为附加存储。顾名思义，就是存储设备通过标准的网络拓扑结构（如以太网）添加到一群计算机上。NAS 是文件级的存储方法，它的重点在于帮助工作组和部门机构解决迅速增加存储容量的需求。如今，用户采用 NAS 大多用于文档、图片和电影共享等，而且随着云计算的发展，一些 NAS 厂商也推出了云存储功能，大大方便了企业和个人用户的使用。

NAS 产品是真正即插即用的产品。NAS 设备一般支持多计算机平台，用户通过网络支持协议可进入相同的文档，因而 NAS 设备无须改造即可用于混合 UNIX/Windows 的局域网内，其应用非常灵活。

但 NAS 有一个关键性问题，即备份过程中消耗网络带宽。与将备份数据流从 LAN（局域网）中转移出去的存储区域网（SAN）不同，NAS 仍使用网络进行备份和恢复。NAS 的一个缺点是它将存储事务由并行 SCSI 连接转移到了网络上，即 LAN 除了处理正常用户的传输流外，还必须处理包括备份操作的存储磁盘请求。

（3）SAN

SAN 通过光纤通道交换机连接存储阵列和服务器主机，最后成为一个专用的存储网络。SAN 经过十多年的发展，已经相当成熟，成为业界的标准（但各个厂商的光纤交换技术不完全相同，其服务器和 SAN 有兼容性的要求）。

笔记

SAN 提供了一种与现有 LAN 连接的简易方法，并且通过同一物理通道支持广泛使用的 IP 和 SCSI 协议。SAN 不受现今主流的、基于 SCSI 存储结构的布局限制。特别重要的是，随着存储容量的爆炸性增长，SAN 允许企业独立地增加它们的存储容量。SAN 的结构允许任何服务器连接到任何存储阵列，这样不管数据置放在哪里，服务器都可直接存取所需的数据，而且因为采用了光纤接口，SAN 还具有更高的带宽。

如今的 SAN 解决方案通常会采取光纤信道（FC SAN）和基于 iSCSI（Internet Small Computer System Interface，一种基于 TCP/IP 的协议）的 SAN（IP SAN）两种形式。光纤信道是 SAN 解决方案中人们最熟悉的类型，但最近一段时间以来，基于 iSCSI 的 SAN 解决方案开始大量出现在市场上，与光纤通道技术相比较而言，这种技术具有良好的性能，而且价格低廉。

SAN 真正地综合了 DAS 和 NAS 两种存储解决方案的优势。例如，在一个很好的 SAN 解决方案实现中，用户可以得到一个完全冗余的存储网络，这个存储网络具有不同寻常的扩展性。确切地说，用户不仅可以得到只有 NAS 存储解决方案才能得到的几百 TB 的存储空间，而且还可以得到只能在 DAS 解决方案中才能得到的块级数据访问功能。从数据访问角度来说，用户还可以得到一个合理的速度，对于那些需要大量磁盘访问的操作来说，SAN 具有更好的性能支持。利用 SAN 解决方案，开发者还可以实

笔记

现存储的集中管理，从而能够充分利用那些处于空闲状态的空间。更具优势的是，在某些实现中，开发者甚至可以将服务器配置为没有内部存储空间的服务器，要求所有的系统都直接从 SAN（只能在光纤通道模式下实现）引导，这也是一种即插即用技术。

SAN 确实具有这些伟大的优点。那么，SAN 的缺陷在哪里？SAN 存在两个较大的缺陷：成本和复杂性，特别是在光纤信道中这些缺陷尤其明显。在使用光纤信道的情况下，合理的成本是 1 TB 或 2 TB 大约就需要 5 万~6 万美元。从另一个角度来看，虽然新推出的基于 iSCSI 的 SAN 解决方案大约只需要 2 万~3 万美元，但是其性能却无法与光纤信道相比。在价格上的差别主要是由于 iSCSI 技术使用的是现在已经大量生产的吉比特以太网硬件，而光纤通道技术要求价格昂贵的特定设备。

因为 SAN 解决方案是从基本功能中剥离出存储功能，所以运行备份操作就无须考虑它们对网络总体性能的影响。SAN 方案也使得管理和集中控制得到简化，特别是在全部存储设备都聚集在一起时。最后一点，光纤接口提供了 10 km 的连接长度，这使得实现物理上的分离、非数据中心的集中存储变得非常容易。

综上所述，DAS 存储一般应用在中小企业，与计算机采用直连方式。NAS 则通过以太网添加到计算机上。SAN 则使用 FC（光纤）接口，提供性能更佳的存储。

如今，随着移动计算时代的来临，更多的非结构化数据产生，这对 NAS 和 SAN 都是一个挑战，NAS+SAN 将是未来主要的存储解决方案，也是目前比较热门的统一存储方式之一。既然是一个集中化的磁盘阵列，那么就得支持主机系统通过 IP 网络进行文件级别的数据访问，或者通过光纤协议在 SAN 网络进行块级别的数据访问。这种磁盘阵列配置了多个存储控制器和一个管理接口，允许存储管理员按需创建存储池或空间，并将其提供给不同访问类型的主机系统。

注意： 统一存储系统的前端主机接口可支持 FC 8 Gbit/s、iSCSI 1 Gbit/s 和 iSCSI 10 Gbit/s，后端具备 SAS 6 Gbit/s 硬盘扩展接口，可支持 SAS、SATA 硬盘及 SSD 固态硬盘接口，具备极佳的扩展能力。实现 FC SAN 与 IP SAN、各类存储介质的完美融合，有效地整合用户现有存储网络架构，实现高性能 SAN 网络的统一部署和集中管理，以适应业务和应用变化的动态需求。主机接口及硬盘接口均采用模块化设计，更换主机接口或硬盘扩展接口，无须更换固件，可大大简化升级维护的难度和工作量。

以下是归纳的集中存储的应用场景：

① DAS 虽然比较古老，但还是适用于那些数据量不大，对磁盘访问速度要求较高的中小企业。

② NAS 多适用于文件服务器，用来存储非结构化数据，虽然受限于以太网的速度，但是部署灵活，成本低廉。

③ SAN 多适用于大型应用或数据库系统，缺点是成本高、较复杂。

2. NFS 服务

NFS（Network File System，网络文件系统）是 FreeBSD（类 UNIX 操作系统）支

持的文件系统中的一种。它允许网络中的计算机之间通过 TCP/IP 网络共享资源。在 NFS 的应用中，本地 NFS 的客户端应用可以透明地读写位于远端 NFS 服务器上的文件，就如同访问本地文件一样。

以下是 NFS 的优点。

① 节省本地存储空间。将常用的数据存放在一台 NFS 服务器上且可以通过网络访问，那么本地终端将可减少自身存储空间的使用。

② 用户不需要在网络中的每台机器上都建有 Home 目录。Home 目录可以放在 NFS 服务器上且可以在网络上被访问和使用。

③ 一些存储设备，如软驱、CDROM 和 Zip（一种高储存密度的磁盘驱动器与磁盘）等，都可以在网络上被别的机器使用。这可以减少整个网络上可移动介质设备的数量。

笔记

3. CIFS（Samba）服务

SMB（Server Message Block）又称 CIFS（Common Internet File System），是由微软公司开发的一种软件程序级的网络传输协议，而 Samba 是在 Linux 和 UNIX 系统上实现 SMB 协议的一个免费软件，由服务器及客户端程序构成。SMB 是一种在局域网上共享文件和打印机的一种通信协议，它为局域网内的不同计算机之间提供文件及打印机等资源的共享服务。SMB 协议是客户机/服务器型协议，客户机通过该协议可以访问服务器上的共享文件系统、打印机及其他资源。通过设置“NetBIOS over TCP/IP”使得 Samba 不但能与局域网络主机分享资源，还能与全世界的计算机分享资源。

Samba 的主要功能是可以用于 Linux 与 Windows 系统间的直接文件共享和打印共享。Samba 既可用于 Windows 与 Linux 之间的文件共享，也可用于 Linux 与 Linux 之间的资源共享。

任务实施

1. 构建 NAS

（1）CentOS 6.5 安装配置 NFS

实验环境：使用 VirtualBox 创建 2 台 CentOS 6.5 的虚拟机，用来搭建 NFS。

IP 地址规划如下。

nfs-server IP：172.24.3.26。

nfs-client IP：172.24.3.30。

① 使用 2 个 YUM 节点安装 NFS 软件包。

```
[root@nfs-server ~]# yum -y install nfs-utils rpcbind
[root@nfs-client ~]# yum -y install nfs-utils rpcbind
```

先在 Server 端创建一个目录，用来共享。

```
[root@nfs-server ~]# mkdir -p /opt/test
[root@nfs-server ~]# ll /opt/
total 0
drwxr-xr-x 2 root root 6 May 11 09:16 test
```

编辑 NFS 的配置文件/etc/exports。

```
# vi /etc/exports
# 在文件中添加一行如下
/opt/test 172.24.3.0/24(rw,no_root_squash,no_all_squash,sync,anonuid=501,anongid=501)
```

② 生效配置。

```
[root@nfs-server ~]# exportfs -r
```

笔记

配置文件说明：

/opt/test：共享目录。若没有这个目录，请新建一个。

172.24.3.0/24：可以为一个网段或一个 IP，也可以是域名。域名支持通配符，如 *.qq.com。

rw：read-write，可读写。

ro：read-only，只读。

sync：文件同时写入硬盘和内存。

async：文件暂存于内存，而不是直接写入内存。

no_root_squash：NFS 客户端连接服务端时，如果使用的是 root，那么对服务端共享的目录来说，也拥有 root 权限。显然，开启该项是不安全的。

root_squash：NFS 客户端连接服务端时，如果使用的是 root，那么对服务端共享的目录来说，拥有匿名用户权限，通常它将使用 nobody 或 nfsnobody 身份。

all_squash：不论 NFS 客户端连接服务端时使用什么用户，对服务端共享的目录来说，都拥有匿名用户权限。

anonuid：匿名用户的 UID（User Identification，用户身份证明）值，可以在此处自行设定。

anongid：匿名用户的 GID（Group Identification，共享资源系统使用者的群体身份）值。

③ Server 端启动 NFS 服务，并设置开机自启。

```
[root@nfs-server ~]#service rpcbind start
[root@nfs-server ~]#service nfs start
[root@nfs-server ~]#chkconfig rpcbind on
[root@nfs-server ~]#chkconfig nfs on
```

④ Server 端查看可挂载目录。

```
[root@nfs-server ~]# showmount -e 172.24.3.26
Export list for 172.24.3.26:
/opt/test 172.24.3.0/24
```

⑤ Client 操作。客户端挂载（客户端挂载前需要关闭两台服务器的防火墙 iptables）。

```
[root@nfs-client ~]# mount -t nfs 172.24.3.26:/opt/test /mnt/
```

无提示即为成功，查看一下挂载情况。

```
[root@nfs-client ~]# df -h
Filesystem              Size  Used Avail Use% Mounted on
/dev/vda1               20G   994M  20G   5%  /
devtmpfs                984M  0     984M  0%  /dev
tmpfs                   1001M 0     1001M 0%  /dev/shm
tmpfs                   1001M 25M   977M  3%  /run
tmpfs                   1001M 0     1001M 0%  /sys/fs/cgroup
tmpfs                   201M  0     201M  0%  /run/user/0
172.24.3.26:/opt/test   20G   1.1G  19G   6%  /mnt
```

⑥ 验证 NFS 共享存储。

在客户端/mnt 下创建一个 abc.txt 的文件进行测试。

```
[root@nfs-client mnt]# touch abc.txt
[root@nfs-client mnt]# ll
total 0
-rw-r--r-- 1 root root 0 May 11 10:31 abc.txt
```

用 md 5 工具计算 abc.txt。

```
[root@nfs-client mnt]# md5sum abc.txt
d41d8cd98f00b204e9800998ecf8427e  abc.txt
```

回到 Server 节点，查看/opt/test 目录。

```
[root@nfs-server ~]# ll /opt/test/
total 0
-rw-r--r-- 1 root root 0 May 11 10:31 abc.txt
```

可以查看到一个 abc.txt 文件，此时再用 md 5 工具计算 abc.txt。

```
[root@nfs-server ~]# md5sum /opt/test/abc.txt
d41d8cd98f00b204e9800998ecf8427e  /opt/test/abc.txt
```

发现两个 md 5 的值是一样的，即验证成功。

⑦ 指定 NFS 协议。NFS 默认使用 UDP，本任务也可换成 TCP。

```
[root@ nfs-client mnt]#mount -t nfs 172.24.3.26:/opt/test /mnt -o proto=tcp -o nolock
```

（2）CentOS 6.5 安装配置 CIFS（Samba）

试验环境：一台 CentOS 6.5 的虚拟机，一台操作系统为 Windows 7 的普通台式机。

CentOS 6.5 的虚拟机就用刚才创建的 nfs-server。

① 主机名修改。

```
[root@ nfs-server ~]# hostname samba
[root@ nfs-server ~]# logout
```

重新连接虚拟机。

```
[root@ samba ~]#
```

Windows 7 机器使用本机作为客户端。

Samba 服务端 IP：172.24.3.26。

② 关闭 SELinux（Security-Enhanced Linux，扩张强制访问控制安全模块）。

```
[root@ samba ~]# setenforce 0
setenforce:SELinux is disabled
```

关闭 Iptables（IP 信息包过滤系统）。

```
[root@ samba ~]# service iptables stop
```

③ 安装和配置 Samba 服务。

```
[root@ samba ~]# yum install samba -y
```

配置 Samba 的配置文件。

```
[root@ samba ~]# vi /etc/samba/smb.conf
```

在最后加入一段代码。

```
[share]
        comment=share
        path=/opt/share
        #这是需要共享出去的目录路径,如果没有该目录就创建一个
        browseable=yes
        public=yes
        writable=yes
```

确保/opt/share 目录的存在，并把该目录的权限设置为 777。

```
[root@samba ~]# chmod 777 /opt/share/
```

④ 创建 Samba 用户。

```
[root@samba ~]# smbpasswd -a root      #该用户必须是系统存在的用户
New SMB password:
Retype new SMB password:
Added user root.
```

为了方便，本任务使用的是 root 用户。输入“smbpasswd -a root”后再输入密码。本任务设置的密码为 000000。

⑤ 重启 Samba 服务。

```
[root@samba ~]# service smb restart
```

⑥ 使用 Windows 访问 NFS。按〈WIN+R〉组合键，在打开的对话框中输入“\\172.24.3.26”，如图 2-2 所示。输入完后单击“确定”按钮。

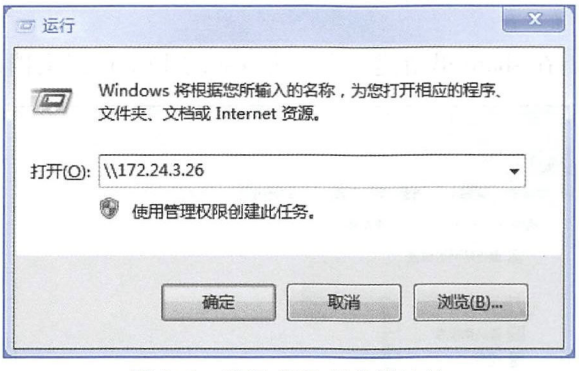


图 2-2　访问 SMB 服务器地址

笔记

在打开的“Windows 安全”对话框中输入用户名“root”，密码“000000”，如图 2-3所示，然后单击“确定”按钮。如果没有打开“Windows 安全”对话框，需要把 NFS 服务停掉。

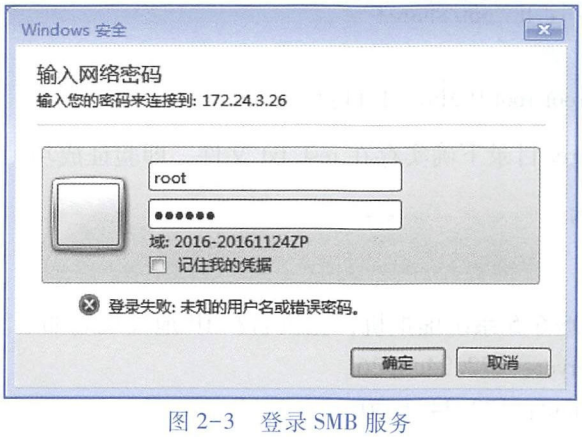


图 2-3　登录 SMB 服务

```
[root@samba ~]# service nfs stop
[root@samba ~]# service rpcbind stop
Warning:Stopping rpcbind.service,but it can still be activated by:
  rpcbind.socket
```

登录后可以看见 share 这个共享目录，如图 2-4 所示。

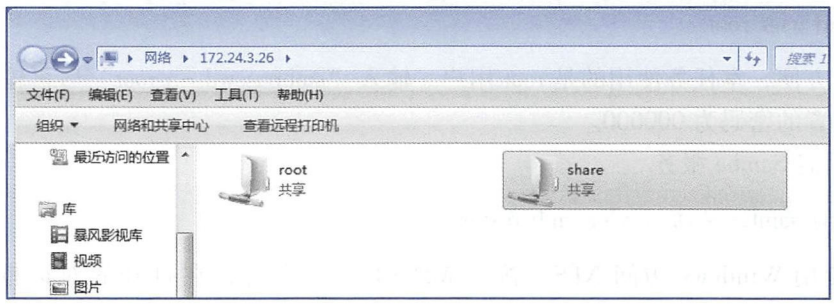


图 2-4 查看共享目录

⑦ 验证 Samba。在 share 里新建一个名为 test 的 TXT 格式文档，如图 2-5 所示。

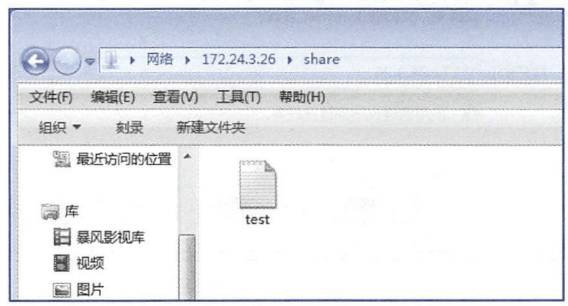


图 2-5 新建文档

然后在服务器上查看。

```
[root@samba ~]# ll /opt/share/
total 0
-rwxr--r-- 1 root root 0 May 11 11:13 test.txt
```

如果在/opt/share/目录下确实存在 test.txt 文件，即验证成功。

2. 构建 IP SAN

实验环境：

创建两台 CentOS 6.5 系统虚拟机，主机名和 IP 地址规划如下。

服务端 iscsi-server：172.24.3.26。

客户端 iscsi-client：172.24.3.30。

修改系统主机名。

```
[root@iscsi-server ~]# hostname iscsi-server
[root@iscsi-client ~]# hostname iscsi-client
```

① 服务端安装。在 Server 端安装 scsi-target-utils 服务。

```
[root@iscsi-server ~]# yum install scsi-target-utils
```

这样就成功地安装了 scsi-target-utils 服务。安装完服务，在配置文件/etc/tgt/targets.conf 的最后加入如下代码。

```
[root@iscsi-server ~]# vi /etc/tgt/targets.conf
<target iqn.2017-04.com.example:target1>    #target 目标名自定义
    backing-store /dev/vdb3                 #scsi 指定用的磁盘或者分区,可自行分区
    initiator-address 172.24.3.0/24         #发起程序可访问的网段
    write-cache off                         #关闭写入缓存
</target>
```

重启服务。

```
[root@iscsi-server ~]#service tgtd restart
```

查看 iscsi target。

```
[root@iscsi-server ~]# tgt-admin -show
Target 1:iqn.2017-04.com.example:target1
    System information:
        Driver:iscsi
        State:ready
    I_T nexus information:
    LUN information:
        LUN:0
            Type:controller
            SCSI ID:IET     00010000
            SCSI SN:beaf10
            Size:0 MB,Block size:1
            Online:Yes
            Removable media:No
            Prevent removal:No
            Readonly:No
            SWP:No
            Thin-provisioning:No
```

```
            Backing store type:null
            Backing store path:None
            Backing store flags:
        LUN:1
            Type:disk
            SCSI ID:IET     00010001
            SCSI SN:beaf11
            Size:8590 MB,Block size:512
            Online:Yes
            Removable media:No
            Prevent removal:No
            Readonly:No
            SWP:No
            Thin-provisioning:No
            Backing store type:rdwr
            Backing store path:/dev/vdb3
            Backing store flags:
    Account information:
    ACL information:
        172.24.3.0/24
```

笔记

其中，LUN：0 是控制器；LUN：1 是上述步骤中实现的共享设备。至此，iSCSI Target 配置完毕。接下来开始配置客户端。

② 客户端安装。

安装 iscsi-initiator-utils 服务。

```
[root@iscsi-client ~]# yum -y install iscsi-initiator-utils
```

用客户端寻找 iscsi Target。

```
[root@iscsi-clent ~]# iscsiadm --mode discovery -t sendtargets -p 172.24.3.26:3260
Starting iscsid:                                           [  OK  ]
172.24.3.26:3260,1 iqn.2017-04.com.example:target1
```

在客户端找到并发现这个设备后，客户端连接此设备。

```
[root@iscsi-clent ~]# iscsiadm -m node -T iqn.2017-04.com.example:target1 -p 172.24.3.26 --login
Logging in to [iface:default,target:iqn.2017-04.com.example:target1,portal:172.24.3.26,3260](multiple)
```

```
Login to [iface: default, target: iqn.2017 - 04.com.example: target1, portal:
172.24.3.26,3260] successful.
```

使用“fdisk-l”命令查看。

```
[root@iscsi-clent ~]# fdisk -l

Disk /dev/vda:21.5 GB,21474836480 bytes,41943040 sectors
Units=sectors of 1 * 512=512 bytes
Sector size (logical/physical):512 bytes/512 bytes
I/O size (minimum/optimal):512 bytes/512 bytes
Disk label type:dos
Disk identifier:0x000af71d

Device Boot      Start     End      Blocks   Id  System
/dev/vda1   *    2048    41929649  20963801  83  Linux

Disk /dev/sda:8589 MB,8589934592 bytes,16777216 sectors
Units=sectors of 1 * 512=512 bytes
Sector size (logical/physical):512 bytes/512 bytes
I/O size (minimum/optimal):512 bytes/512 bytes
```

可以看见一块大小为 8 GB 的/dev/sda 设备，首先对这块设备进行分区，最后把所有的空间都分给/dev/sda1。

```
[root@iscsi-clent ~]# fdisk -l

Disk /dev/vda:21.5 GB,21474836480 bytes,41943040 sectors
Units=sectors of 1 * 512=512 bytes
Sector size (logical/physical):512 bytes/512 bytes
I/O size (minimum/optimal):512 bytes/512 bytes
Disk label type:dos
Disk identifier:0x000af71d

Device Boot      Start     End      Blocks   Id  System
/dev/vda1   *    2048    41929649  20963801  83  Linux

Disk /dev/sda:8589 MB,8589934592 bytes,16777216 sectors
Units=sectors of 1 * 512=512 bytes
Sector size (logical/physical):512 bytes/512 bytes
```

```
I/O size (minimum/optimal):512 bytes/512 bytes
Disk label type:dos
Disk identifier:0x78ec7ca6

Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048    16777215     8387584   83  Linux
```

对分区进行格式化。

```
[root@iscsi-clent ~]# mkfs.ext4 /dev/sda1
mke2fs 1.42.9 (28-Dec-2013)
Filesystem label=
OS type:Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks,Stripe width=0 blocks
524288 inodes,2096896 blocks
104844 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=2147483648
64 block groups
32768 blocks per group,32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768,98304,163840,229376,294912,819200,884736,1605632

Allocating group tables:done
Writing inode tables:done
Creating journal (32768 blocks):done
Writing superblocks and filesystem accounting information:done
```

格式化完成后，对该设备进行挂载。

```
[root@iscsi-clent ~]# mount /dev/sda1 /mnt/
[root@iscsi-clent ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        20G  979M   20G   5% /
devtmpfs        984M     0  984M   0% /dev
tmpfs          1001M     0 1001M   0% /dev/shm
tmpfs          1001M   17M  985M   2% /run
```

```
tmpfs           1001M   0     1001M   0%   /sys/fs/cgroup
/dev/sda1       7.8G    36M   7.3G    1%   /mnt
```

从挂载情况可以看出/dev/sda1 挂载在/mnt 上，/dev/sda1 设备和本地的设备无异，验证 iSCSI Target 完毕。

项目实训

【实训题目】

在 CentOS 6.5 操作系统中构建 iSCSI。

【实训目的】

1. 掌握 iSCSI 的工作原理。
2. 掌握 iSCSI 的部署方式。
3. 了解 iSCSI 外置存储的优点。

【实训内容】

1. 在服务端安装配置 iscsi-target-utils 服务，提供大小为 10 GB、名称为 testvg-test-lv 的逻辑卷。
2. 查看 iSCSI Target 信息，显示控制器和共享的设备。
3. 在客户端安装 iscsi-initiator-utils 服务，利用客户端寻找服务端 iSCSI Target。
4. 对共享的设备进行分区，分区大小为 5 GB，并进行格式化。
5. 将格式化的分区挂载至/opt/test 中。

任务 2.2　构建 Openfiler 存储系统

任务描述

1. 了解存储虚拟化技术。
2. 了解 Openfiler 网络存储管理系统。
3. 了解 NAS 和 iSCSI 的工作原理。
4. 使用 Openfiler 构建 NAS 和 IP SAN。

知识学习

1. 存储虚拟化

存储虚拟化（Storage Virtualization），简单地说就是通过对底层的存储硬件资源进行抽象化而展现出来的一种逻辑表现。它通过将实体存储空间（如硬盘）进行逻辑分隔，组成不同的逻辑存储空间。它通过一个逻辑存储实体代表底层复杂的物理驱动器，屏蔽掉了单个存储设备的容量、速度等物理特性，而且也屏蔽了底层驱动器的复

笔记

杂性以及存储系统后端拓扑结构的多样性，从而极大地增强了数据的存储能力、可恢复性和性能表现。

对服务器和应用程序来说，通过存储虚拟化技术，无论后端的物理存储是什么，面对它们的都是存储设备的逻辑映像。对于用户来说，他们所面对的是一种抽象的物理磁盘，它们与存储资源中大量的物理特性隔离开来。用户不必去关心实际的后端存储，只需要专注于管理存储空间本身，用户所看到的逻辑存储单元和本地的硬盘没有什么差别。

因此，存储虚拟化技术与传统技术相比，它具有更少的运营费用和更低的复杂性。它简化了物理存储设备的配置和管理任务，同时还能够充分利用现有的物理存储资源，避免了存储资源的浪费。

2. Openfiler

Openfiler 是一个定制化的、基于 Web 方式进行存储管理的网络存储操作系统。Openfiler 在单一框架中提供了基于文件的网络附加存储（Network Attached Storage）和基于块的存储区域网络（Storage Area Network）存储解决方案。Openfiler 由 rPath Linux 驱动，是由 GNU General Public License version 2（GNU 通用公共许可证第 2 版）进行授权的一种自由软件。同时，它的软件接口是基于使用开放源代码的第三方软件，如 Apache、Samba、LVM 2、Ext 3、Linux NFS 和 iSCSI Enterprise Target 等。Openfiler 通过将这些开源技术整合到一个统一的架构体系架构中，以一个基于 Web 方式功能强大的管理界面来提供虚拟存储功能。

Openfiler 的强大之处在于其对多种网络协议的支持。这些网络协议包括 NFS、iSCSI、SMB/CIFS、FTP 和 HTTP/WebDAV 等。对于网络目录的支持，Openfiler 包括了 NIS（Network Information Services 网络信息服务）、Active Directory（活动目录）、LDAP（Lightweight Directory Access Protocol，轻量目录访问协议）、基于 Windows NT 的域控制器和 Hesiod。对于认证协议的支持，它支持 Kerberos 5（一种安全认证系统）的认证协议。对于分区技术的支持，Openfiler 支持基于卷的分区技术，如本地文件系统的 Ext3、XFS 和 JFS（Journal File System，日志文件系统）格式，以及实时快照和相应的磁盘配额管理。它通过统一标准的接口，使基于各种网络文件系统协议的共享资源分配变得更容易、快捷与高效。

Openfiler 所提供的强大的虚拟存储功能特性，使其在以动态、灵活、可伸缩为特性的云计算环境中，成为了非常有力的云端存储解决方案之一。在生产环境中，它大大降低了花费在网络存储硬件设备上的部署和维护成本。同时它还可以作为实验科研环境下的一种高效、低成本的网络存储模拟解决方案，具有很高的实用价值优势。本单元任务中，将使用一个基于 Openfiler 的虚拟镜像，通过服务的配置和管理，向读者展示一个基于自己云端环境需求的虚拟存储解决方案的产生过程。

3. NAS

NAS 是一种网络存储器，也是一种专用数据存储服务器。它包括存储器件（如磁盘阵列、CD/DVD 驱动器、磁带驱动器或可移动的存储介质）和内嵌系统软件，可提

供跨平台文件共享功能。NAS 通常在一个 LAN 上占有自己的节点，无须应用服务器的干预，允许用户在网络上存取数据。在这种配置中，NAS 集中管理和处理网络上的所有数据，将负载从应用或企业服务器上卸载下来，有效降低成本。

NAS 本身能够支持多种协议（如 NFS、CIFS、FTP、HTTP 等），而且能够支持各种操作系统。通过任何一台工作站，采用 IE 或 Netscape 浏览器就可以对 NAS 设备进行直观、方便的管理。

4. IP SAN（iSCSI）

（1）iSCSI 的由来

iSCSI 技术是一种由 IBM 公司研究开发的，供硬件设备使用的可以在 IP 协议的上层运行的 SCSI 指令集。这种指令集可以实现在 IP 网络上运行 SCSI 协议，使其能够在诸如高速千兆以太网上进行路由选择。iSCSI 技术是一种新储存技术。该技术是将现有 SCSI 接口与以太网络（Ethernet）技术结合，使服务器可与使用 IP 网络的储存装置互相交换资料。

Internet 小型计算机系统接口（iSCSI）是一种基于 TCP/IP 的协议，用来建立和管理 IP 存储设备、主机和客户机等之间的相互连接，并创建存储区域网络（SAN）。SAN 使得 SCSI 协议应用于高速数据传输网络成为可能，这种传输以数据块级别（Block-Level）在多个数据存储网络间进行。

SCSI 结构基于客户机/服务器模式，其通常应用环境为：当设备互相靠近，并且这些设备由 SCSI 总线连接。iSCSI 的主要功能是在 TCP/IP 网络上的主机系统（启动器 Initiator）和存储设备（目标器 Target）之间进行大量数据的封装和可靠传输过程。此外，iSCSI 还提供了在 IP 网络封装 SCSI 命令，且运行在 TCP 上。

笔记

（2）iSCSI 的工作过程

当 iSCSI 主机应用程序发出数据读写请求后，操作系统会生成一个相应的 SCSI 命令。该 SCSI 命令在 iSCSI Initiator 层被封装成 iSCSI 消息包并通过 TCP/IP 传送到设备侧，设备侧的 iSCSI Target 层会解开 iSCSI 消息包，得到 SCSI 命令的内容，然后传送给 SCSI 设备执行。设备执行 SCSI 命令后的响应，在经过设备侧 iSCSI Target 层时被封装成 iSCSI 响应 PDU（Protocol Data Unit，协议数据单元），通过 TCP/IP 网络传送给主机的 iSCSI Initiator 层，iSCSI Initiator 会从 iSCSI 响应 PDU 里解析出 SCSI 响应并传送给操作系统，最后操作系统再响应给应用程序。

（3）iSCSI 启动器

从本质上说，iSCSI 启动器是一个客户端设备。它连接到服务器提供的某一服务，并发起对该服务的请求。如果利用 iSCSI 创建 Oracle RAC（Oracle Real Application Cluster，真正应用集群），iSCSI 启动器软件需要安装在每个 Oracle RAC 节点上。

iSCSI 启动器可以用软件实现，也可以用硬件实现。软件 iSCSI 启动器可用于大部分主要操作系统平台，可以使用 iscsi-initiator-utils RPM 中提供的免费 Linux Open-iSCSI 软件驱动程序。iSCSI 软件启动器通常与标准网络接口卡（NIC，多数情况下是千兆位以太网卡）配合使用。硬件启动器是一个 iSCSI HBA（或 TCP 卸载引擎卡）。

它本质上只是一个专用以太网卡，其上的 SCSI ASIC 可以从系统 CPU 内卸载所有工作（TCP 和 SCSI 命令）。iSCSI HBA 可以从许多供应商处（包括 Adaptec、Alacritech、Intel 和 QLogic 公司）购买。

任务实施

（1）ESXi 基本环境准备

IP 地址分配如下。

ESXi：192. 168. 1. 129。

ESXi VMkernel：192. 168. 1. 142。

Openfiler：192. 168. 1. 137。

① 安装 ESXi 操作系统。安装主机网卡需要服务器版网卡支持，否则会出现安装失败，如图 2-6 和图 2-7 所示。

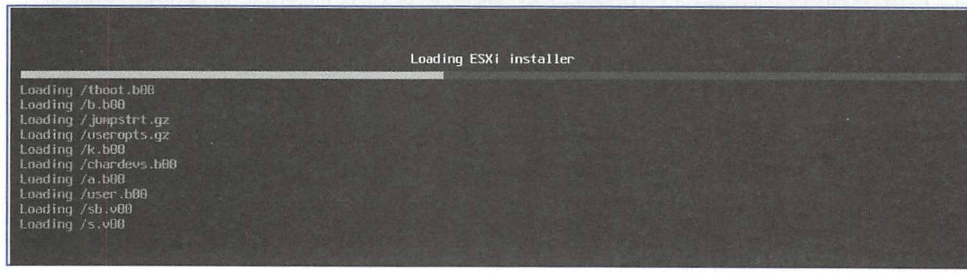


图 2-6 安装 ESXi 操作系统之一

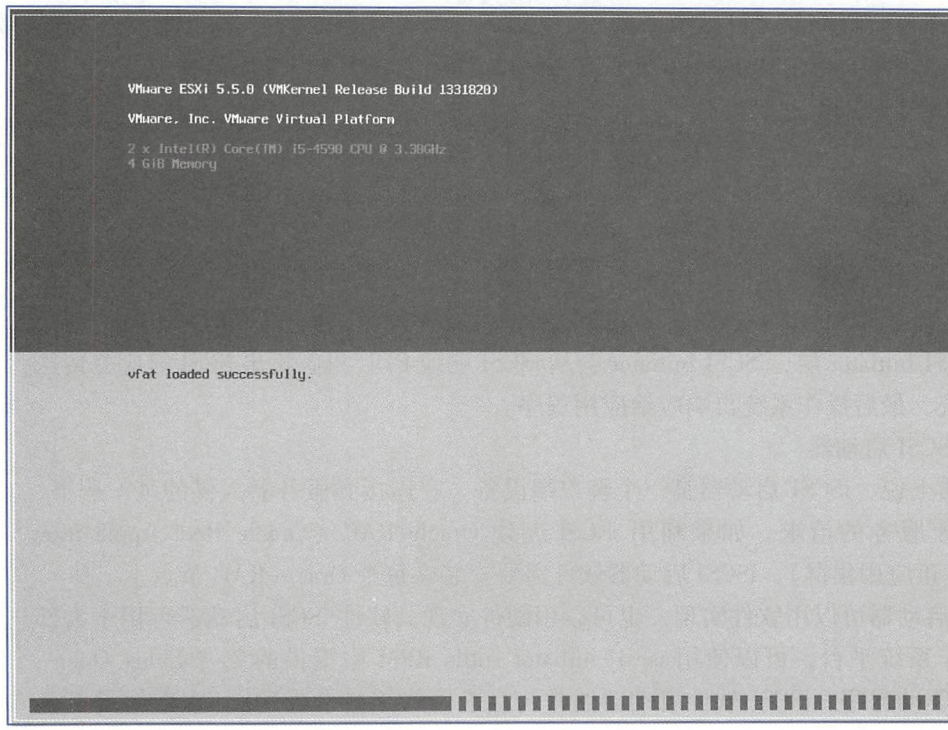


图 2-7 安装 ESXi 操作系统之二

配置 ESXi 系统相关信息。在安装过程中根据提示设置 root 用户密码，如图 2-8 所示。系统安装完成后按〈F2〉键进入，如图 2-9 所示。

图 2-8　设置 root 用户密码

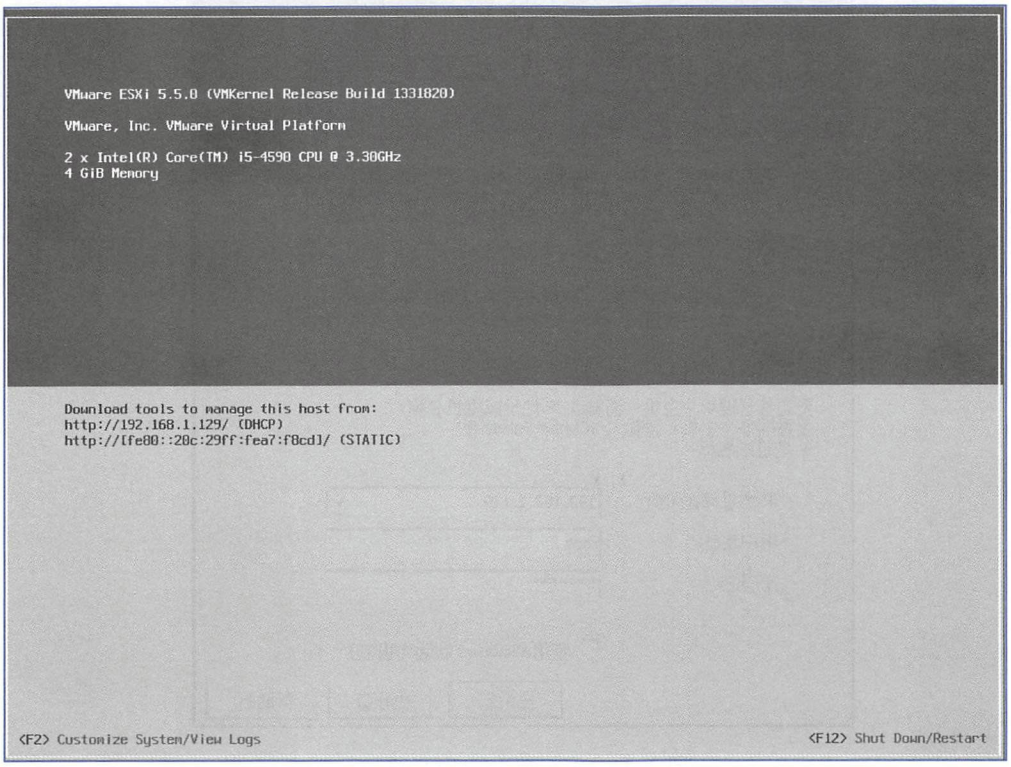


图 2-9　启动 ESXi 操作系统

笔记

② 设置 IP。按要求配置 ESXi 主机地址，本任务安装采用动态获取 IP 地址，即选中“Use dynamic IP address and network configuration”复选项，开发者也可使用静态 IP 地址，如图 2-10 所示。

③ 远程连接 ESXi。使用 VMware vSphere Client 远程连接 ESXi，如图 2-11 所示。

（2）安装 Openfiler 操作系统

本任务安装的是 Openfiler 虚拟机操作系统，在正式的生产环境中，建议在物理机

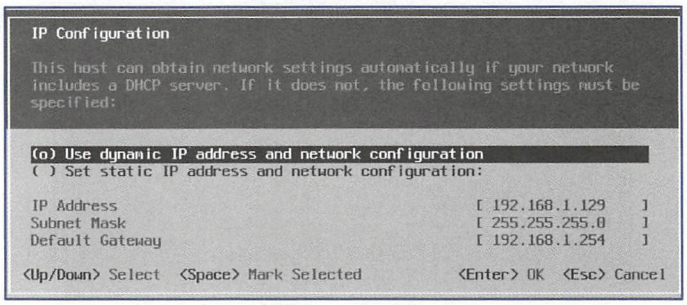


图 2-10 配置 ESXi 地址

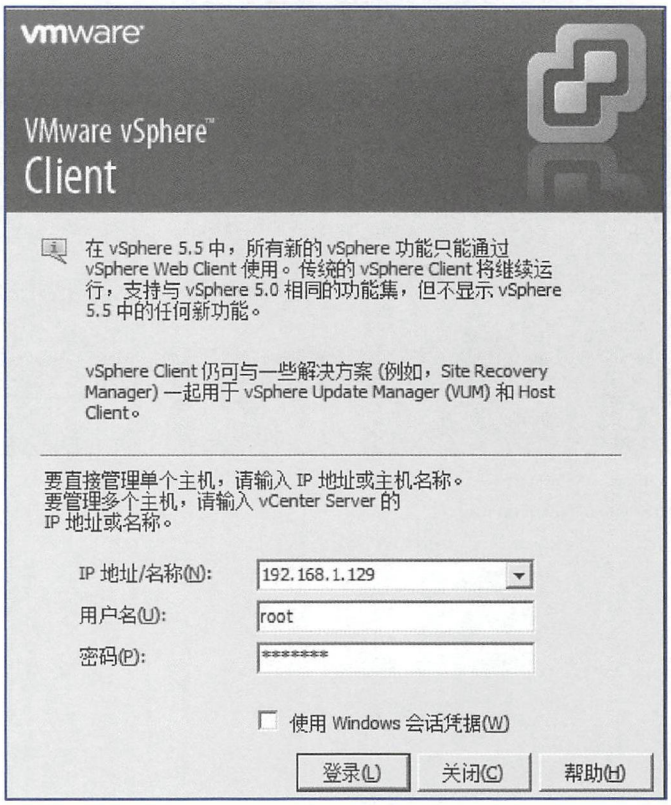


图 2-11 客户端连接 ESXi 系统

中安装 Openfiler 操作系统。

① 上传镜像包。在如图 2-12 所示的界面右侧选择“配置”选项卡，然后选择“硬件”列表的“存储器”选项卡，在“数据存储”列表框中右击“datastore1”标识，在弹出的快捷菜单中选择“浏览数据存储”命令。在打开的“数据存储浏览器”窗口中，单击“上载数据”按钮，上传 Openfiler 镜像包，如图 2-13 所示。

② 配置虚拟机。在如图 2-12 所示界面中单击左侧的 ESXi 服务器（192.168.1.29），在主菜单中选择“文件”→“新建”→“虚拟机”命令。在打开

的“创建新的虚拟机”窗口中按图 2-14 所示设置虚拟机的各项参数，然后不断地单击“下一步”按钮，直到单击“完成”按钮，虚拟机创建完成。

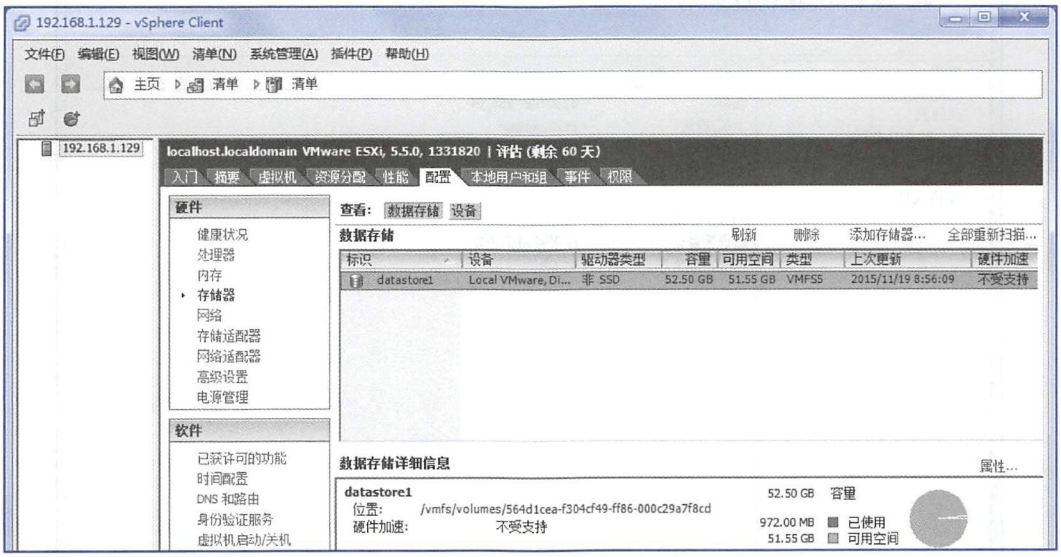


图 2-12　选择存储器

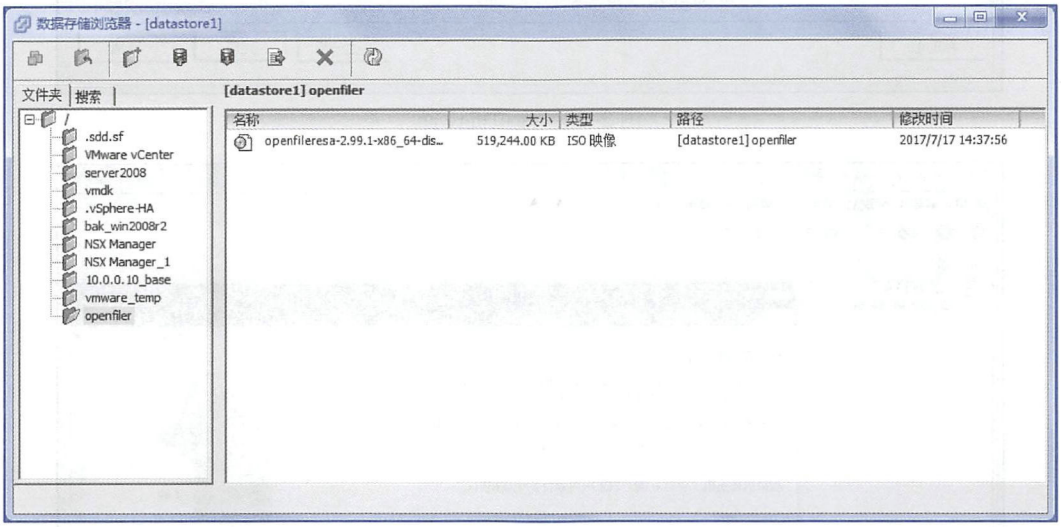


图 2-13　上传 Openfiler 镜像包

单击主界面左侧的 ESXi 服务器（192. 168. 1. 29）下的虚拟机 openfiler-86x_64，在界面右侧“入门”选项卡的“基本任务”列表中，单击“编辑虚拟机设置”按钮，如图 2-15 所示。

在弹出的窗口中，设置虚拟机内存为 2 GB（2048 MB），CPU 内核为 2，镜像（CD/CVD 驱动器 1）使用之前上传的 ISO 文件，“设备状态”组中选中“打开电源时连接”复选框，最后单击“确定”按钮，如图 2-16 所示。

图 2-14 虚拟机设置

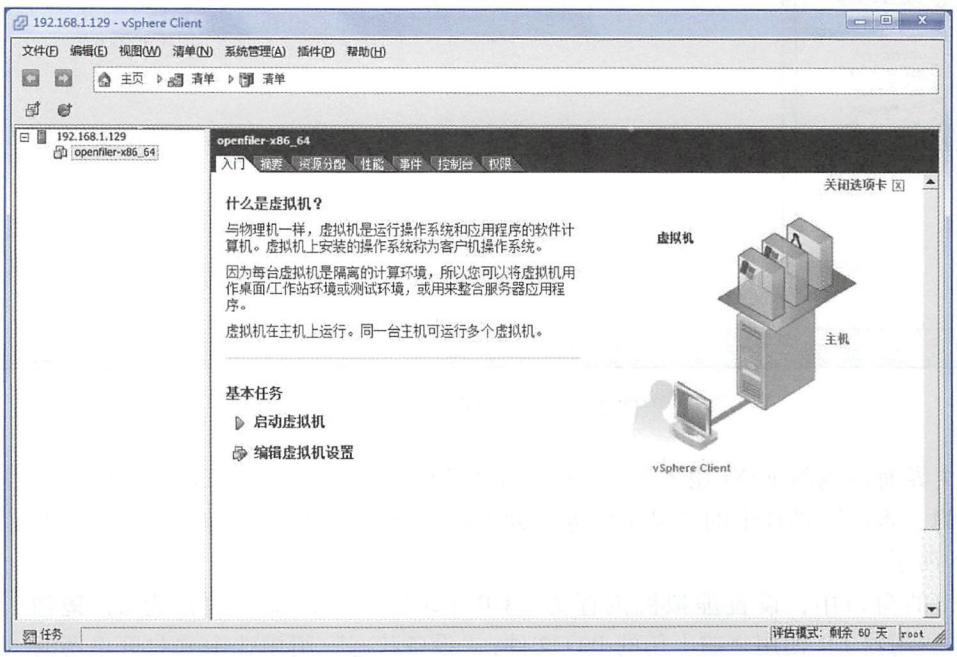


图 2-15 编辑虚拟机设置

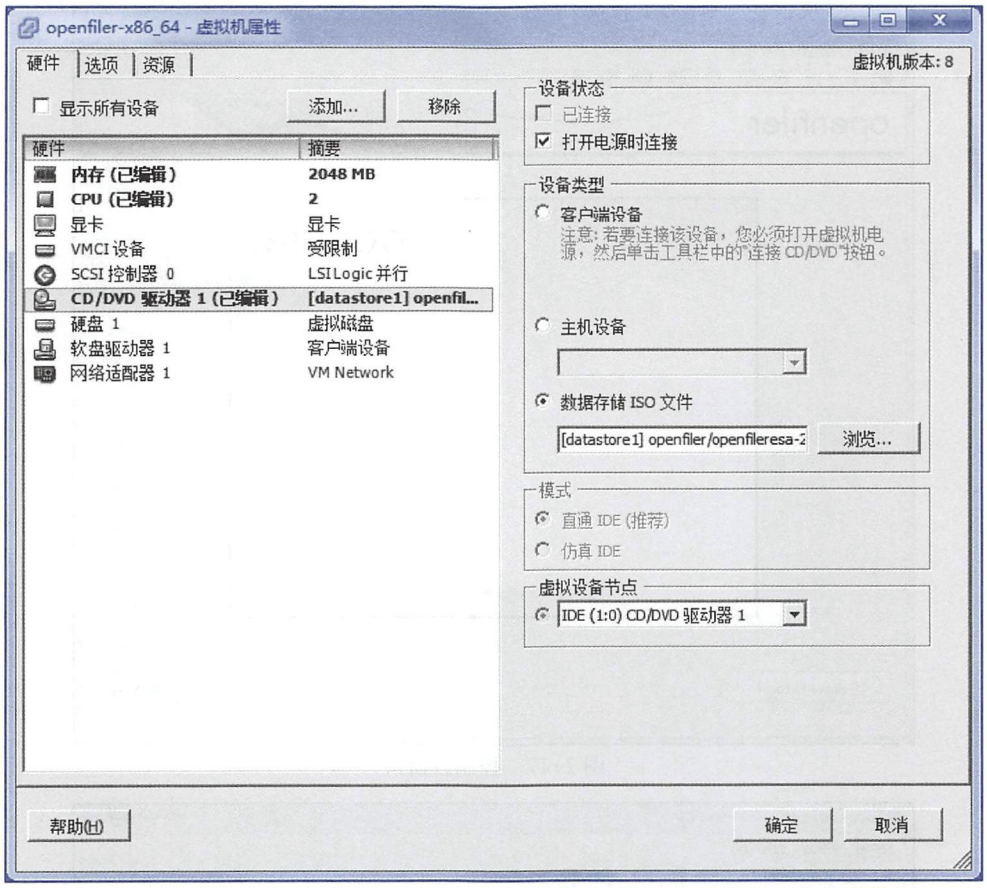


图 2-16　编辑虚拟机属性

笔 记

③ 安装虚拟机。单击主界面左侧的 ESXi 服务器（192.168.1.29）下的虚拟机 openfiler-86x_64，在界面右侧“入门”选项卡的“基本任务”列表中，单击“启动虚拟机”按钮，最后单击“启动虚拟控制台”按钮，弹出的控制台窗口如图 2-17 所示。

此时，进入到的是 Openfiler 安装界面，安装过程进入系统键盘配置界面，在这里选择默认键盘设置，单击“Next”按钮。接下来安装过程进入系统磁盘分区设置。Openfiler 提供了自动磁盘分区和手动磁盘分区两种分区设置方式。在这里推荐手动磁盘分区方式（Create custom layout），如图 2-18 所示。因为这种方式能够根据实际环境自定义配置，更好地满足系统磁盘分区配置需求。此时，系统会询问是否初始化磁盘并删除所有数据，根据需要选择是否保留已有分区或数据。因为本任务此处为空白硬盘，所以选择清除所有数据。系统分区情况如图 2-19 所示。

分配 IP 地址，在“Network Devices”列表中选中“Active on Boot”复选框，使 DHCP 自动获取。然后单击“Next”按钮，如图 2-20 所示。这里也可以根据实际需求手动添加 IP 地址。

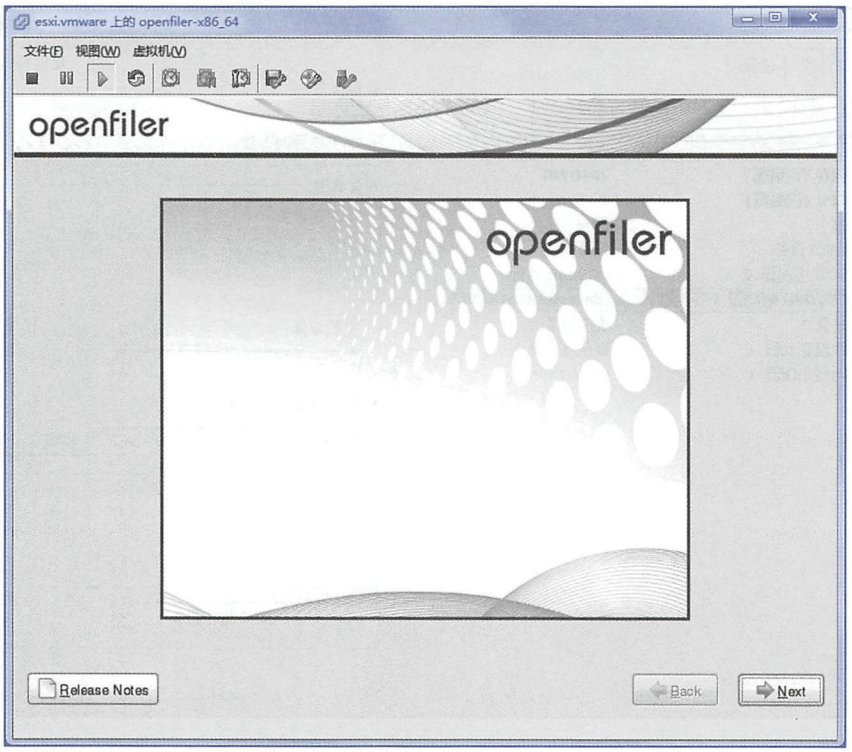


图 2-17 控制台窗口

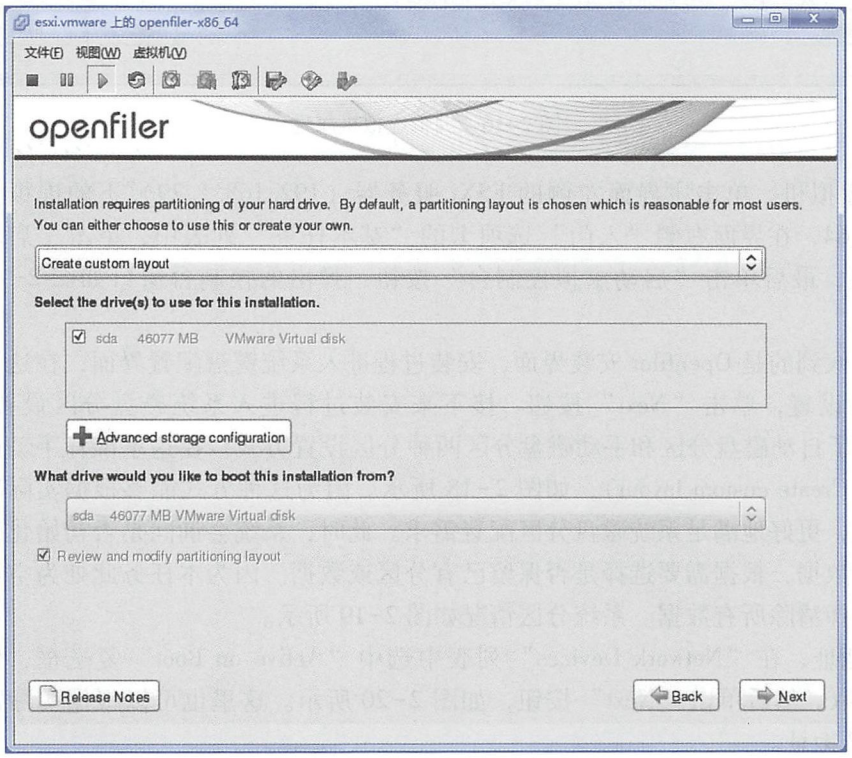


图 2-18 选择手动分区方式

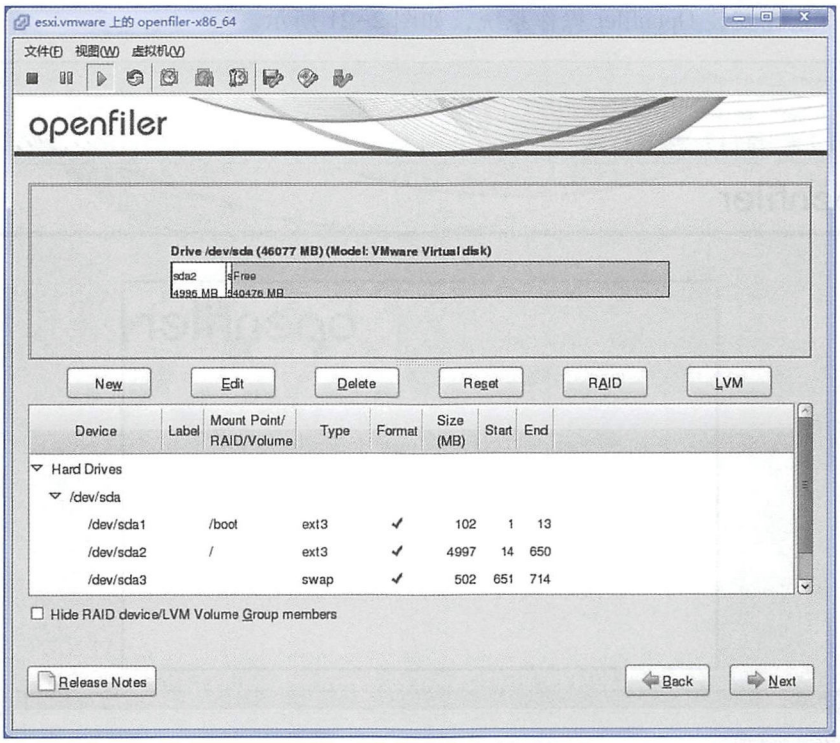


图 2-19　系统分区情况

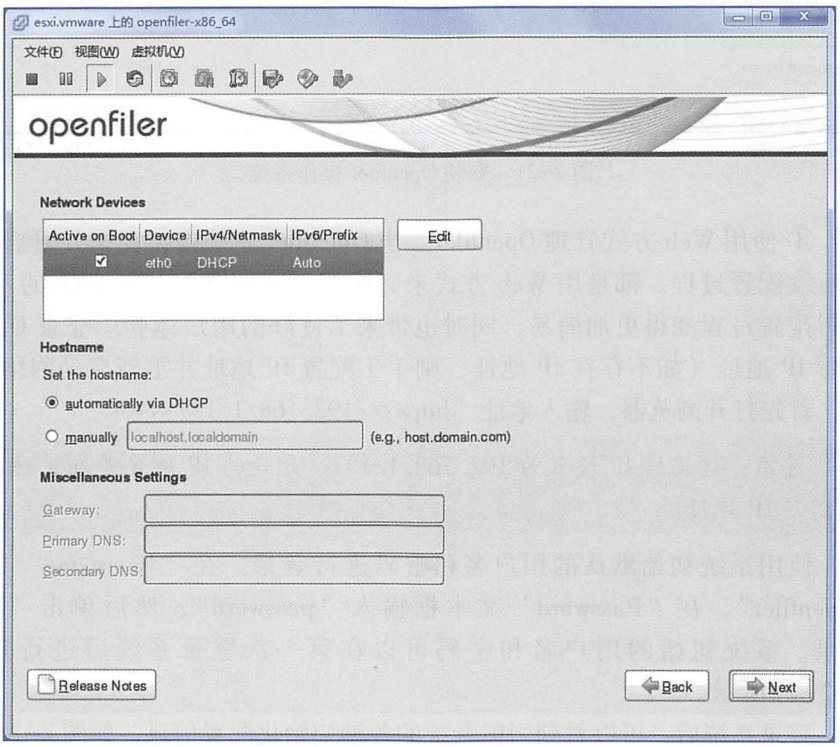
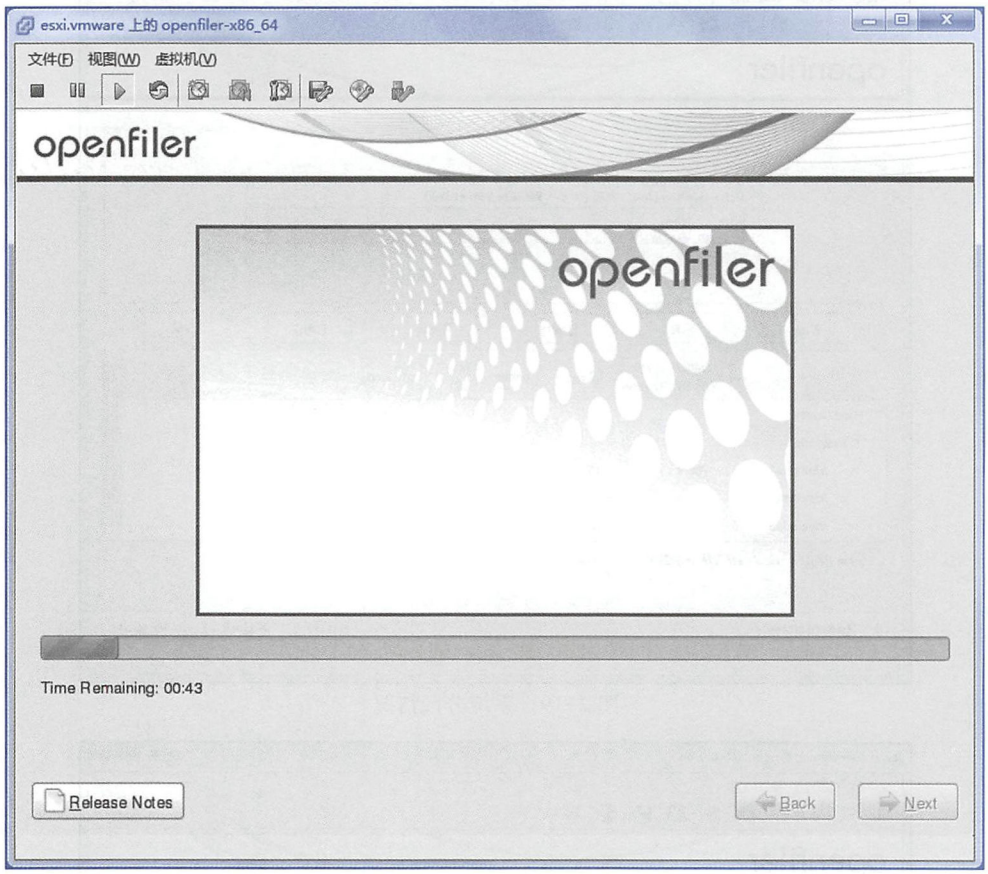


图 2-20　自动获取 DHCP

正在安装 Openfiler 操作系统，如图 2-21 所示。



笔 记

图 2-21 安装 Openfiler 操作系统

④ 使用 Web 方式管理 Openfiler。当 Openfiler 系统安装完毕并重启后，系统所有的后续配置过程，都是用 Web 方式来完成的。这种全部基于 Web 的系统配置方式，使得配置过程变得更加简易，同时也带来了良好的用户体验。登录 Openfiler 虚拟机查看 IP 地址（如不存在 IP 地址，则手工配置 IP 地址并重新启动网络服务）。

首先打开浏览器，输入地址“https://192.168.1.137:446”。

注意：这里的 IP 设置为 192.168.1.137。此处的 IP 地址是在系统安装时所配置的固定 IP 地址。

使用系统初始默认的用户名和密码进行登录。在“Username”文本框中输入“openfiler”，在“Password”文本框输入“password”，然后单击“Log In”按钮登录。系统初始的用户名和密码可以在第一次登录系统后进行重新设置，如图 2-22所示。

登录系统后，可以看到当前系统的各种初始化配置信息，如图 2-23 所示。

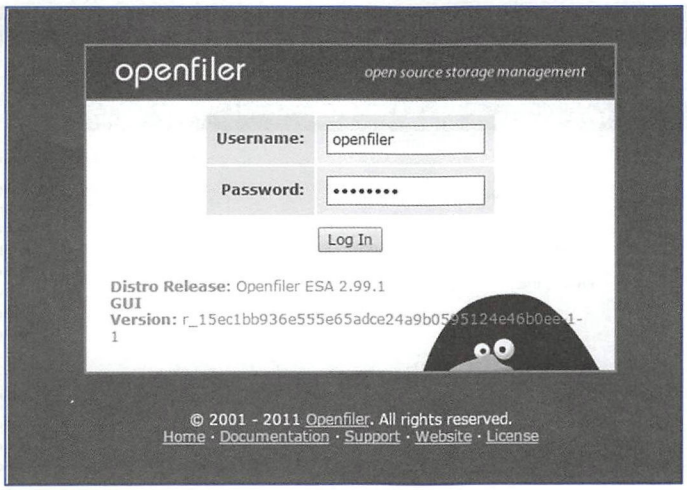


图 2-22　登录 Web 管理界面

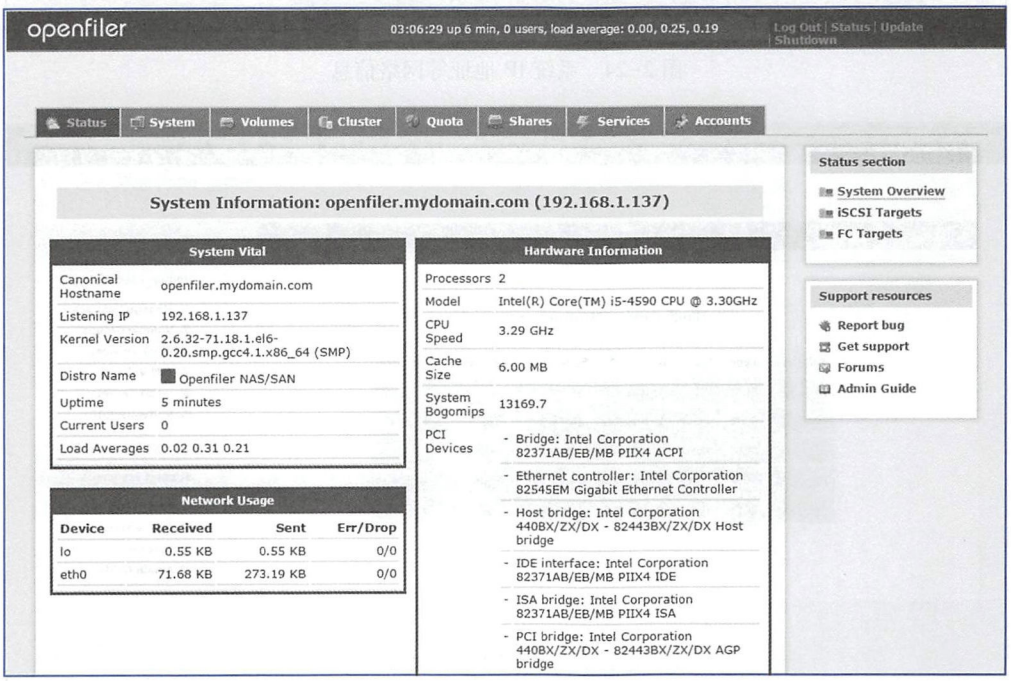


图 2-23　Web 管理界面首页

选择“System”选项卡，可以检查当前系统的 IP 地址等网络信息。如果想对网络进行重新配置，可以在此处修改，如图 2-24 所示。

（3）创建 NAS 卷

① 创建分区。选择“Volumes”选项卡，在界面右侧“Volumes section”列表中选择“Block Devices”选项，这时系统会显示当前所挂载的硬盘信息，如图 2-25 所示。

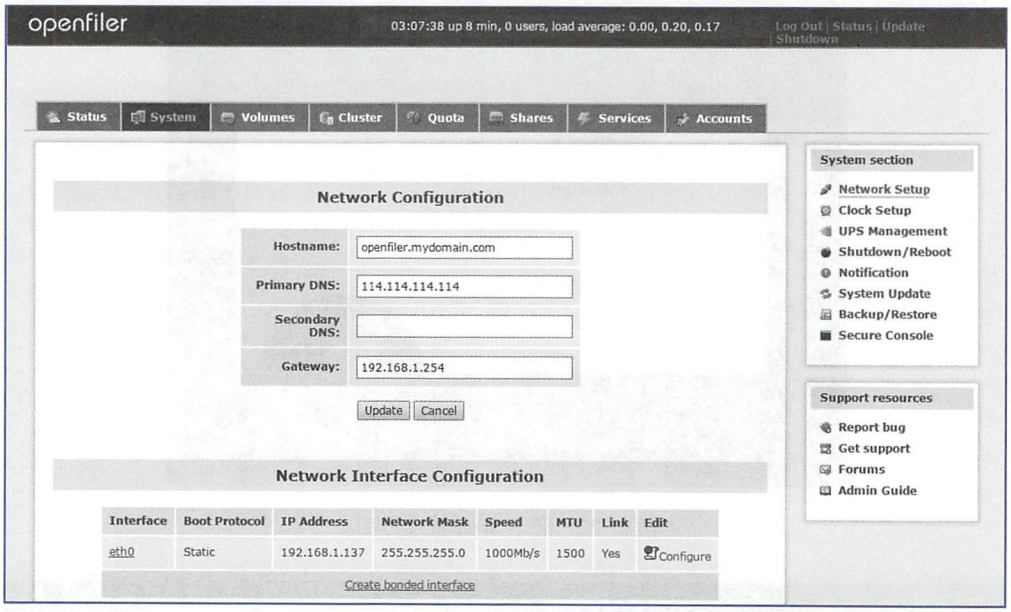


图 2-24 系统 IP 地址等网络信息

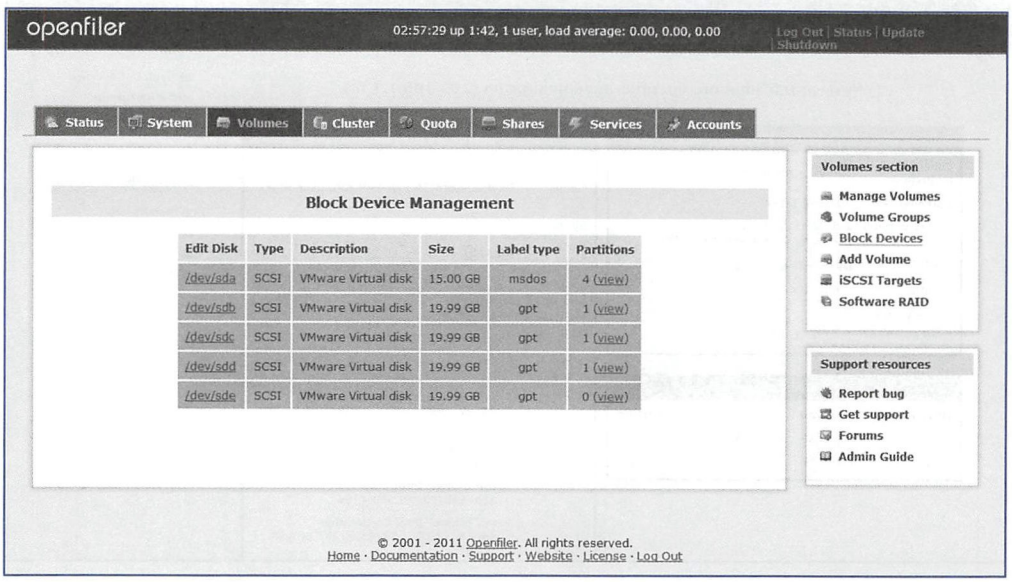


图 2-25 当前所挂载的硬盘信息

单击磁盘/dev/sde 名称，系统会显示当前磁盘的详细分区信息。

要创建一个新的分区。首先在“Create a partition in /dev/sde”列表中单击“Partition Type”下的下拉按钮▼，在弹出的下拉菜单中选择“Physical volume”选项，然后在“Ending cylinder”属性处设置物理卷的大小，本任务选择默认，最后单击“Create”按钮创建，如图 2-26 所示。

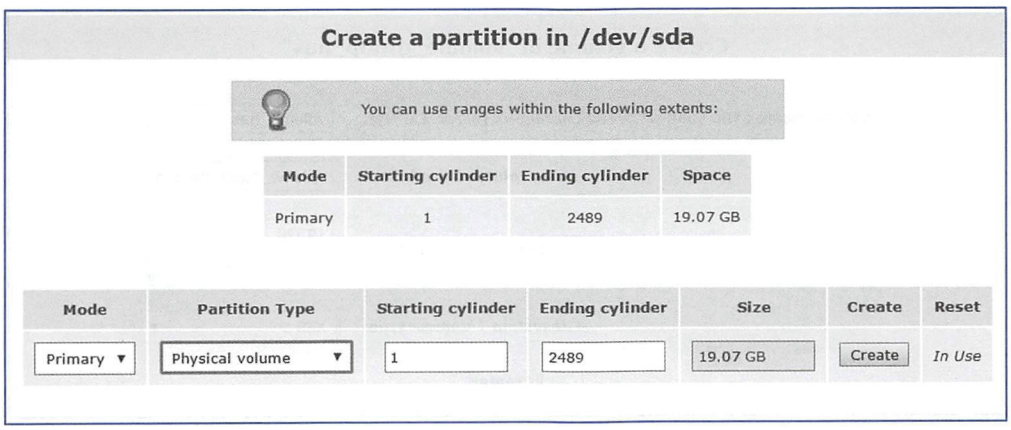


图 2-26　创建新的分区

② 创建卷组。在界面右侧“Volumes section”列表中选择“Volume Groups”选项来创建一个卷组。在“Volume group name”（卷组名称）文本框中输入“volume_group_nas”，同时选中之前所创建的物理卷/dev/sde1，最后单击“Add volume group”按钮，如图 2-27 所示。

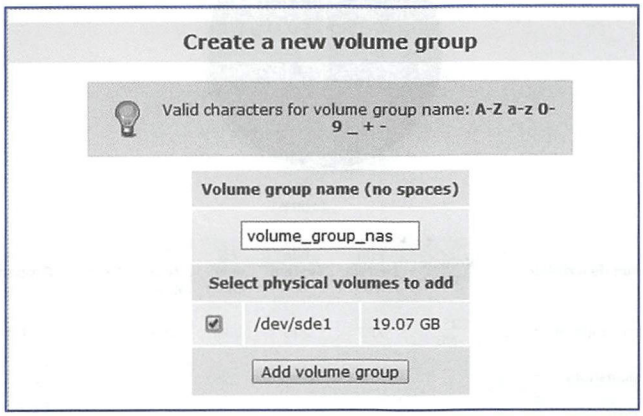


图 2-27　创建新的卷组

笔记

③ 添加卷。在界面右侧“Volumes section”列表中选择“Add Volume”选项，在刚刚创建好的卷组 volume_group_nas 中新建一个卷。在“Volume Name”文本框中输入“volume_nas”，在“Volume Description”文本框中输入“volume_supporting_n”，在“Required Space”文本框中输入“19520”，在“Filesystem / Volume type”下拉列表中选择“XFS”选项，然后单击“Create”按钮，如图 2-28 所示。创建完成后的卷信息如图 2-29 所示。

（4）配置 NAS 存储服务

① 启动 NFS 服务。开启“NFS Server”系统服务。选择“Services”选项卡，然后在界面右侧的“Services section”列表中选择“Manage Services”选项。然后将

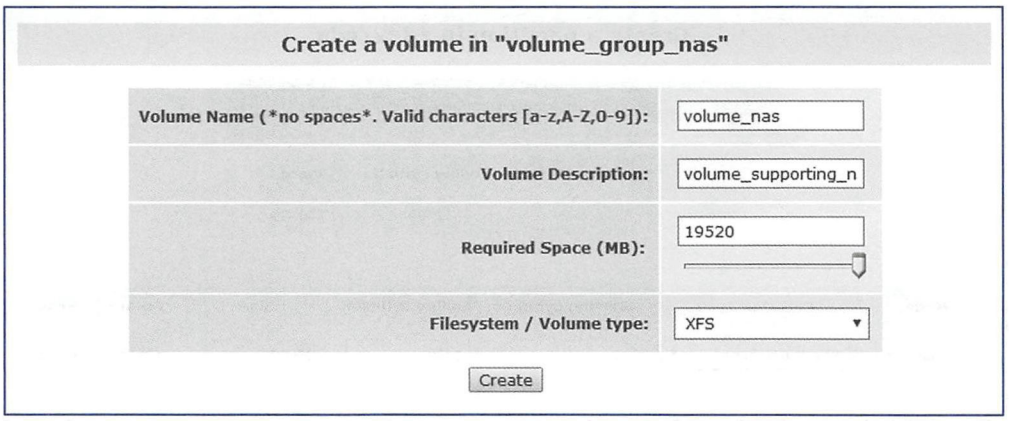


图 2-28 创建卷

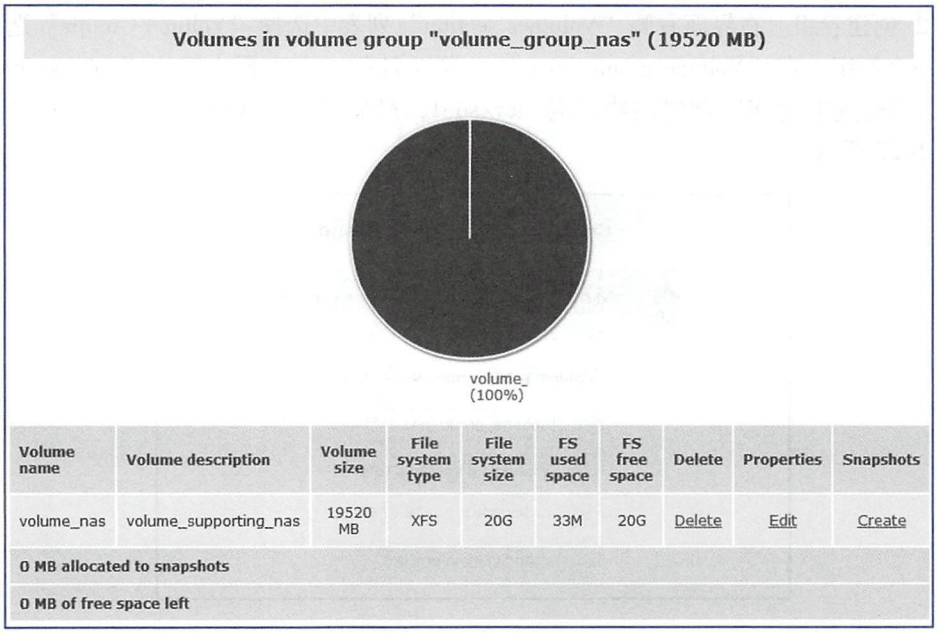


图 2-29 新建卷信息

Manage Services（系统服务）列表中的 NFS Server 的 Boot Status 设置为 Enabled 状态，Current Status 设置为 Running 状态，从而使系统能够对外提供基于 NAS 协议的虚拟存储服务，如图 2-30 所示。

② 编辑共享文件夹。选择“Shares”选项卡，然后在界面右侧的“Shares section”列表中选择“Existing Shares”选项，在跳转的“Network Shares”列表中选择“volume_supporting_nas”选项。然后在弹出的“Folder name”文本框中输入一个共享文件夹的名称（本任务输入的是“nas_folder_shares”），并单击“Create Sub-folder”按钮创建，如图 2-31 所示。

选择刚才创建的共享文件夹，在弹出的对话框左下角单击“Make Share”按钮共

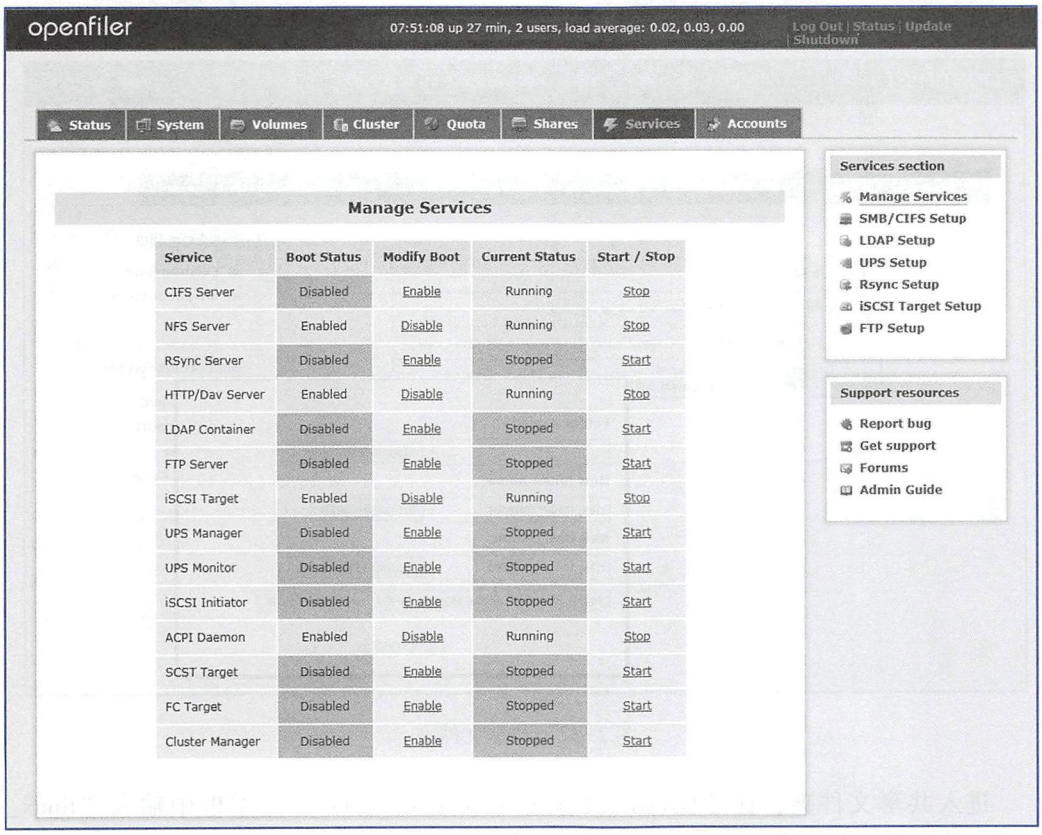


图 2-30　开启 NFS 服务

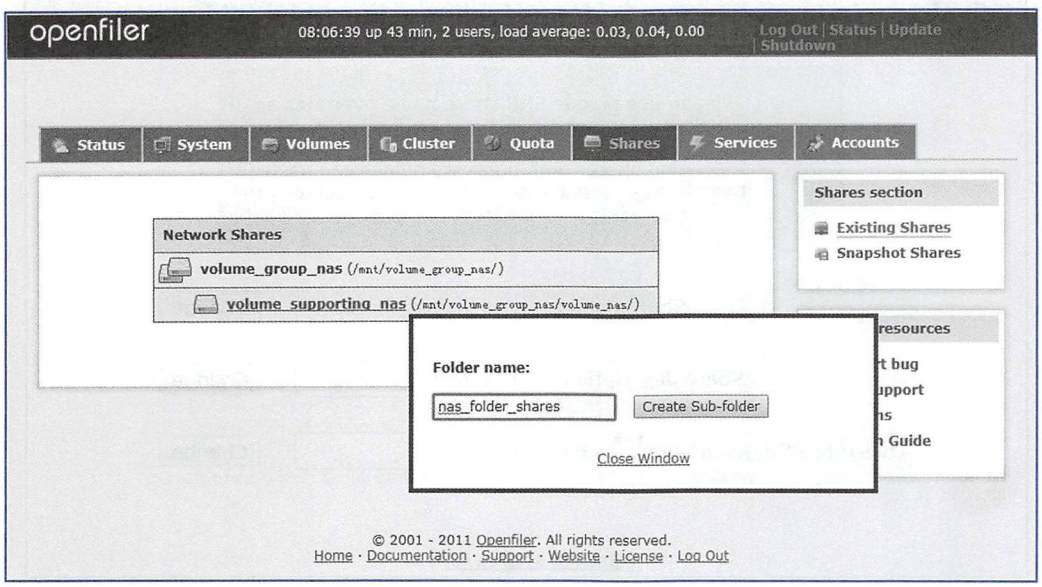


图 2-31　创建共享文件夹

享此文件夹，如图 2-32 所示。

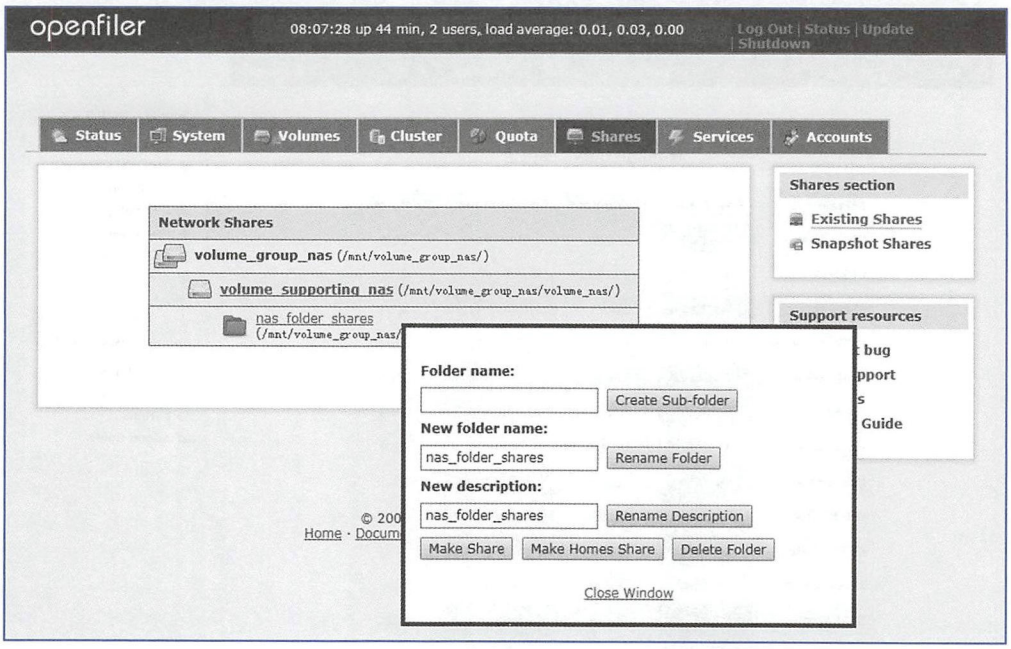


图 2-32 共享文件夹

进入共享文件夹，在“Override SMB/Rsync share name”文本框中输入“linux_test”，给共享文件夹设置别名，如图 2-33 所示。

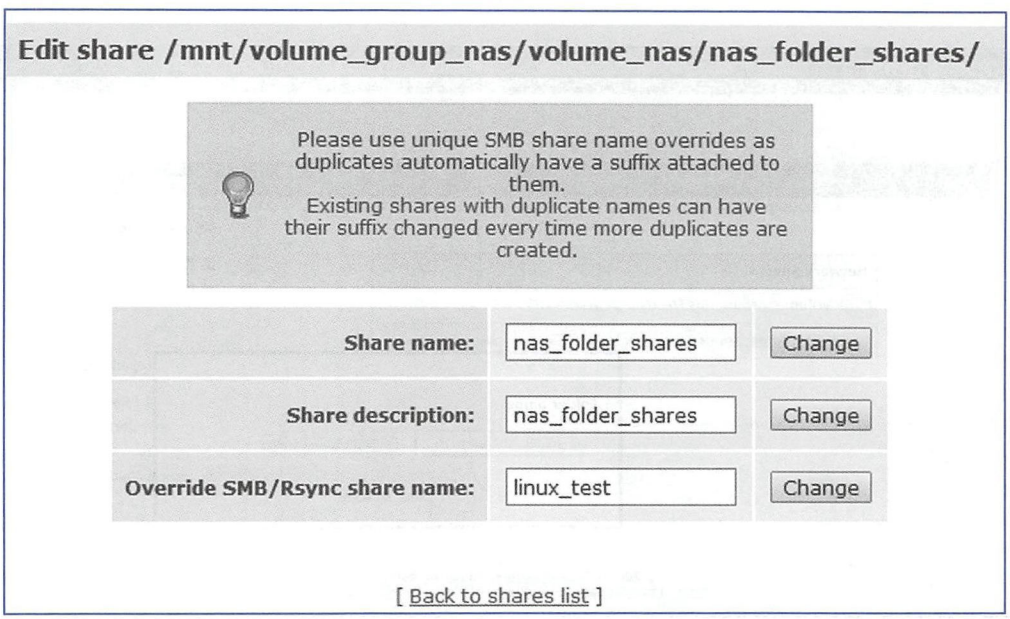


图 2-33 共享文件夹配置别名

笔 记

在“Share Acess Control Mode”列表中设置访问模式为匿名访问，即选中“Public guest access”单选按钮，完成后单击“Update”按钮生效，如图 2-34 所示。

图 2-34　设置访问模式

③ 添加访问控制列表。选择“System”选项卡，在“Network Access Configuration”列表中配置允许访问 Openfiler 系统的安全访问控制列表。在网络访问控制列表中，在“Network/Host”文本框中输入“192.168.200.0”，新增网段地址。然后单击“Type”下的下拉按钮▼，在弹出的下拉列表中选择“Share”选项，添加完成后单击“Update”按钮完成系统配置更新，结果如图 2-35 所示。

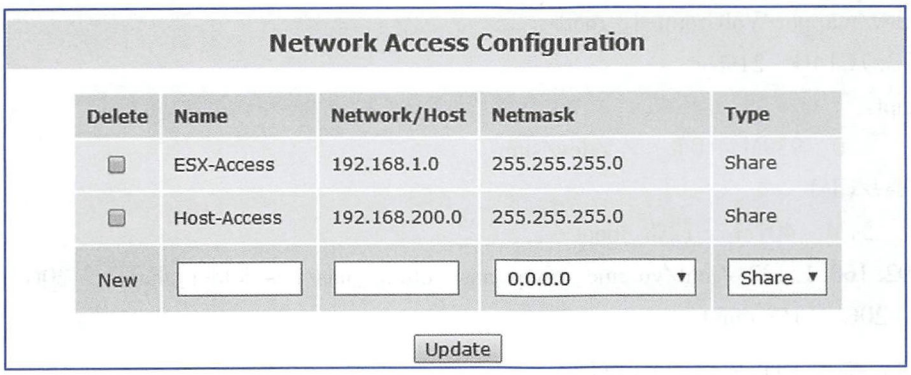


图 2-35　添加访问控制列表

④ 修改服务权限。选择“Shares”选项卡，进入共享文件夹，在“Host access configuration”配置列表中配置外部主机连接的服务权限。这里将 NFS 服务设置为 RW（读写权限）然后单击“Update”按钮生效，如图 2-36 所示。

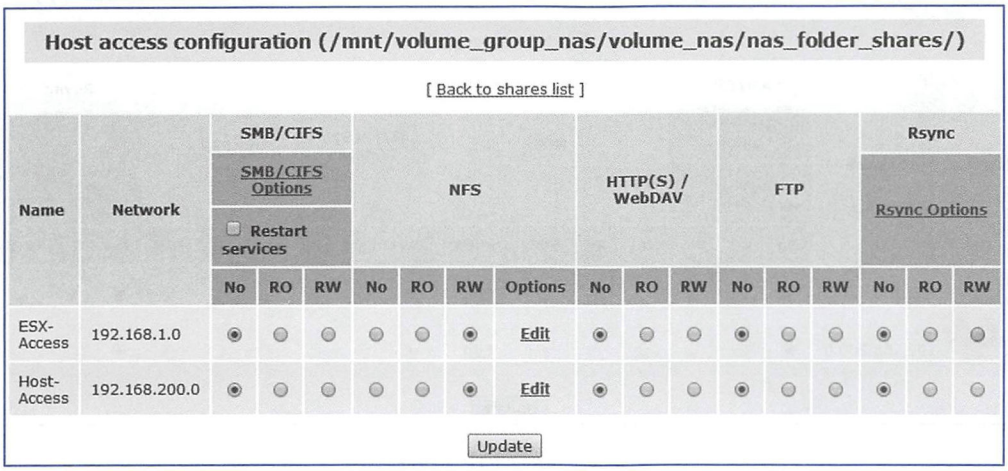


图 2-36　修改 NFS 服务权限

⑤ 利用 NFS 方式验证。登录到其他主机输入命令，可以查看可挂载的 NAS 存储。

```
[root@slaver1 ~]# showmount -e 192.168.1.137
Export list for192.168.1.137:
/mnt/volume_group_nas/volume_nas/nas_folder_shares
192.168.200.0/255.255.255.0,192.168.1.0/255.255.255.0
[root@slaver1~]#mount
192.168.1.137:/mnt/volume_group_nas/volume_nas/nas_folder_shares /mnt
[root@slaver1 ~]# df -h
Filesystem
Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup-lv_root
19G  3.7G14G  21% /
tmpfs
939M     0  939M   0%    /dev/shm
/dev/vda1
477M   51M  401M   12% /boot
192.168.1.137:/mnt/volume_group_nas/volume_nas/nas_folder_shares   20G
32M   20G   1% /mnt
```

（5）配置 FTP 连接 NAS 存储

① 修改服务权限。选择“Shares”选项卡，进入共享文件夹，在“Host access configuration”配置列表中配置外部主机连接的服务权限。这里将 FTP 服务设置为 RW（读写权限），然后单击“Update”按钮生效，如图 2-37 所示。

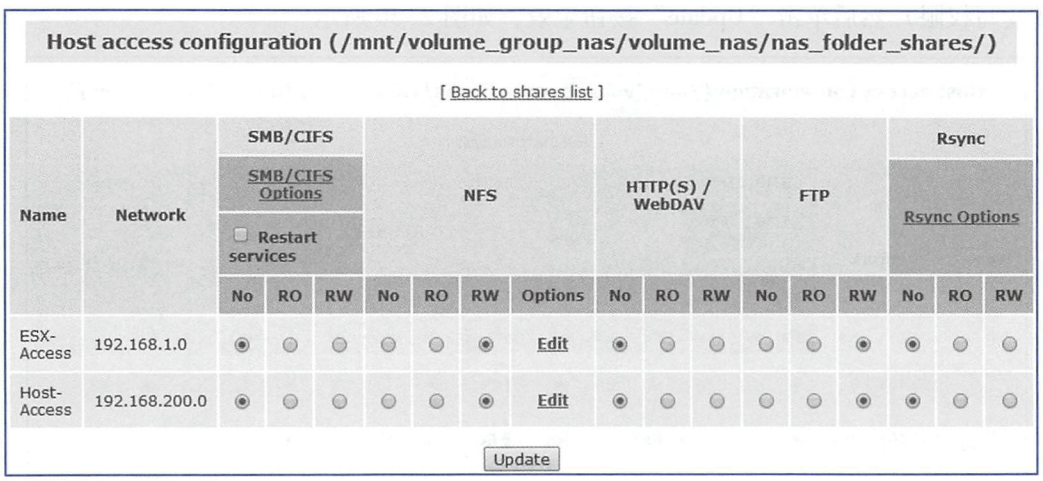


图 2-37 修改 FTP 服务权限

② 启动 FTP 服务。选择“Services”选项卡，在界面右侧“Services section”列表中选择“Manage Services”选项。然后将“Manage Services”（系统服务）列表中的 FTP Server 的 Boot Status 设置为 Enabled 状态，Current Status 设置为 Running 状态，如图 2-38 所示。

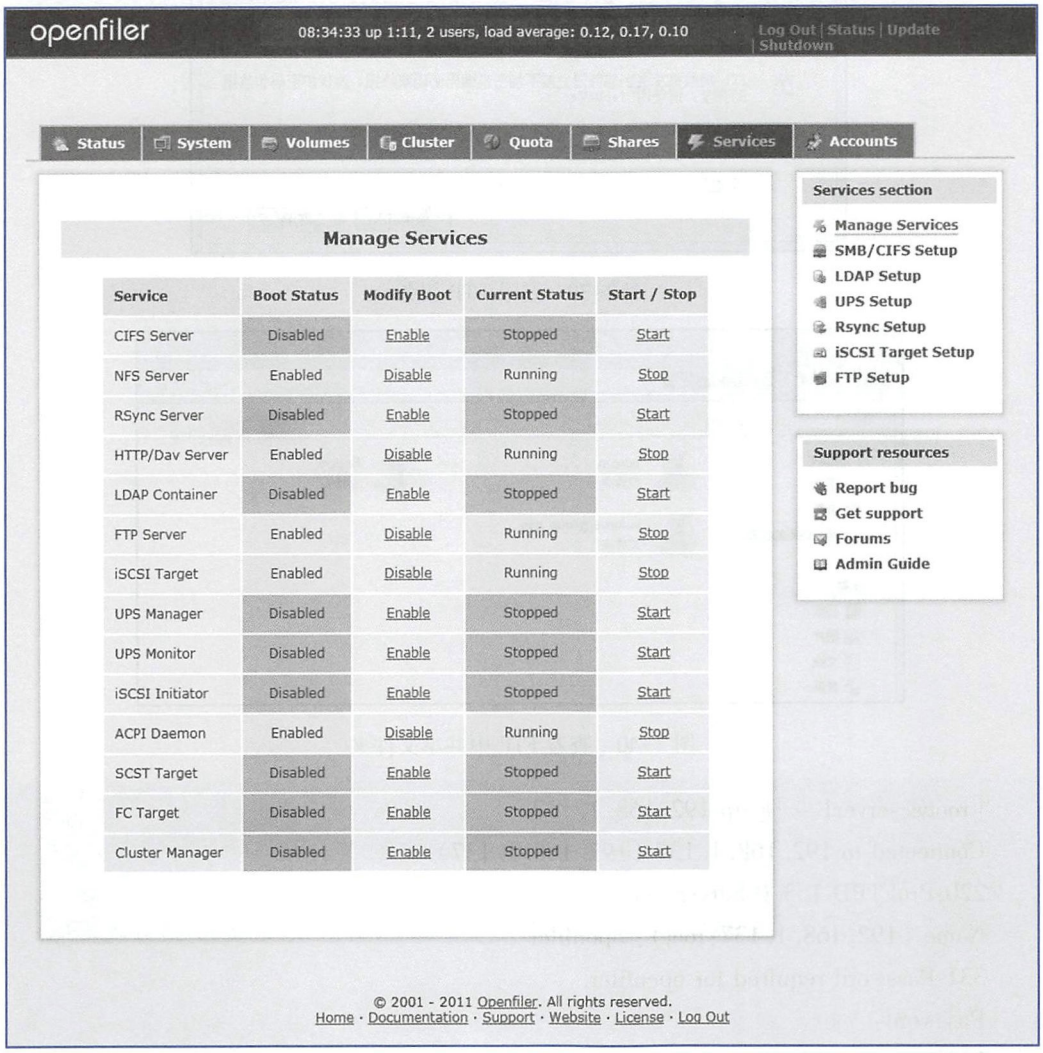


图 2-38　开启 FTP 服务

③ Windows 方式验证，登录 FTP 服务。打开 Windows 资源管理器，在地址栏输入“ftp://192.168.1.137/”，在打开的“身份登录”对话框的“用户名”文本框中输入“openfiler”，在“密码”文本框中输入“password”，如图 2-39 所示。进入服务器，此时可以查看到创建的 NAS 共享文件夹，如图 2-40 所示。

④ 命令行方式验证。

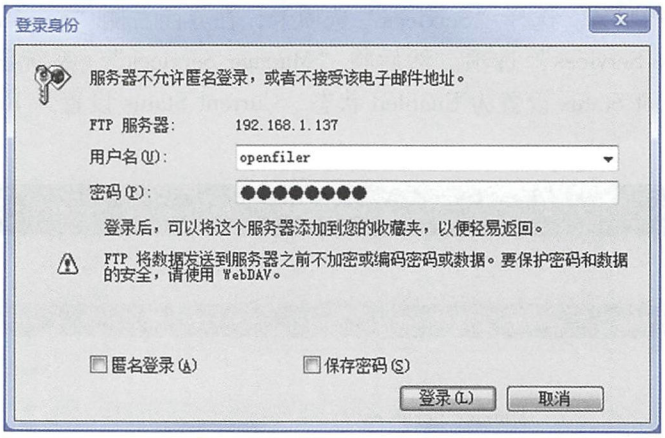


图 2-39 登录 FTP 服务

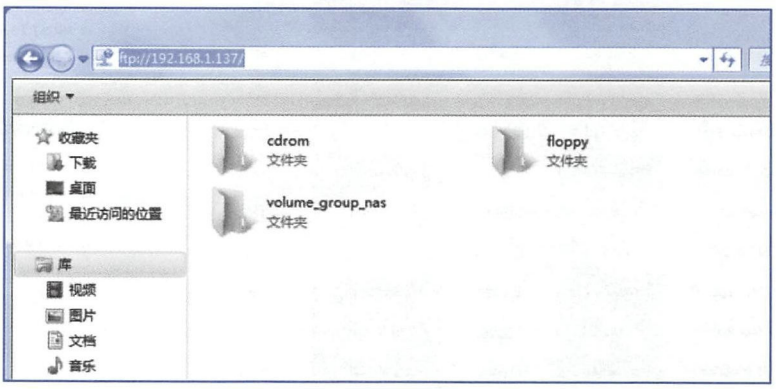


图 2-40 查看 FTP 中共享文件夹

```
[root@server1 ~]# ftp 192.168.1.137
Connected to 192.168.1.137 (192.168.1.137).
220 ProFTPD 1.3.0 Server ready.
Name (192.168.1.137:root): openfiler
331 Password required for openfiler.
Password:
230 User openfiler logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (192.168.1.137,246,207).
150 Opening ASCII mode data connection for file list
drwxr-xr-x   2 root     root         4096 Aug 28   2009 cdrom
drwxr-xr-x   2 root     root         4096 Aug 28   2009 floppy
```

```
drwxr-xr-x    3 root       root            4096 Jul 17 07:04 volume_group_nas
226 Transfer complete.
ftp> cd volume_group_nas
250 CWD command successful
ftp> ls
227 Entering Passive Mode (192.168.1.137,223,223).
150 Opening ASCII mode data connection for file list
drwxr-xr-x    4 root       root             112 Jul 17 07:07 volume_nas
226 Transfer complete.
ftp> cd volume_nas
250 CWD command successful
ftp> ls
227 Entering Passive Mode (192.168.1.137,221,233).
150 Opening ASCII mode data connection for file list
drwxrwxrwx    2 root       root               6 Jul 17 07:04 homes
drwxrwsrwx    2 (?)        (?)                6 Jul 17 07:07 nas_folder_shares
226 Transfer complete.
ftp> cd nas_folder_shares
250 CWD command successful
ftp> ls
227 Entering Passive Mode (192.168.1.137,243,0).
150 Opening ASCII mode data connection for file list
226 Transfer complete.
ftp> put ftp_test ftp_test
local: ftp_test remote: ftp_test
227 Entering Passive Mode (192.168.1.137,235,226).
150 Opening BINARY mode data connection for ftp_test
226 Transfer complete.
ftp> ls
227 Entering Passive Mode (192.168.1.137,237,119).
150 Opening ASCII mode data connection for file list
-rw-rw-rw-    1 (?)        (?)        0 Jul 17 08:29 ftp_test
226 Transfer complete.
ftp>
```

（6）配置 SMB/CIFS 连接 NAS 存储

① 修改服务权限。在“Host access configuration”配置列表中配置外部主机连接的服务权限。将 SMB/CIFS 服务设置为 RW，然后单击“Update”按钮生效，如图 2-41 所示。

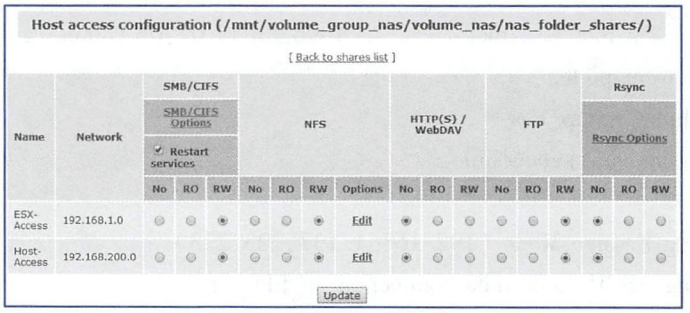


图 2-41 修改 SMB 服务权限

② 启动 CIFS Server 服务。选择“Services”选项卡，在界面右侧“Services section”列表中选择“Manage Services”选项。然后将“Manage Services”列表中的 CIFS Server 的 Boot Status 设置为 Enabled 状态，Current Status 设置为 Running 状态，如图 2-42 所示。

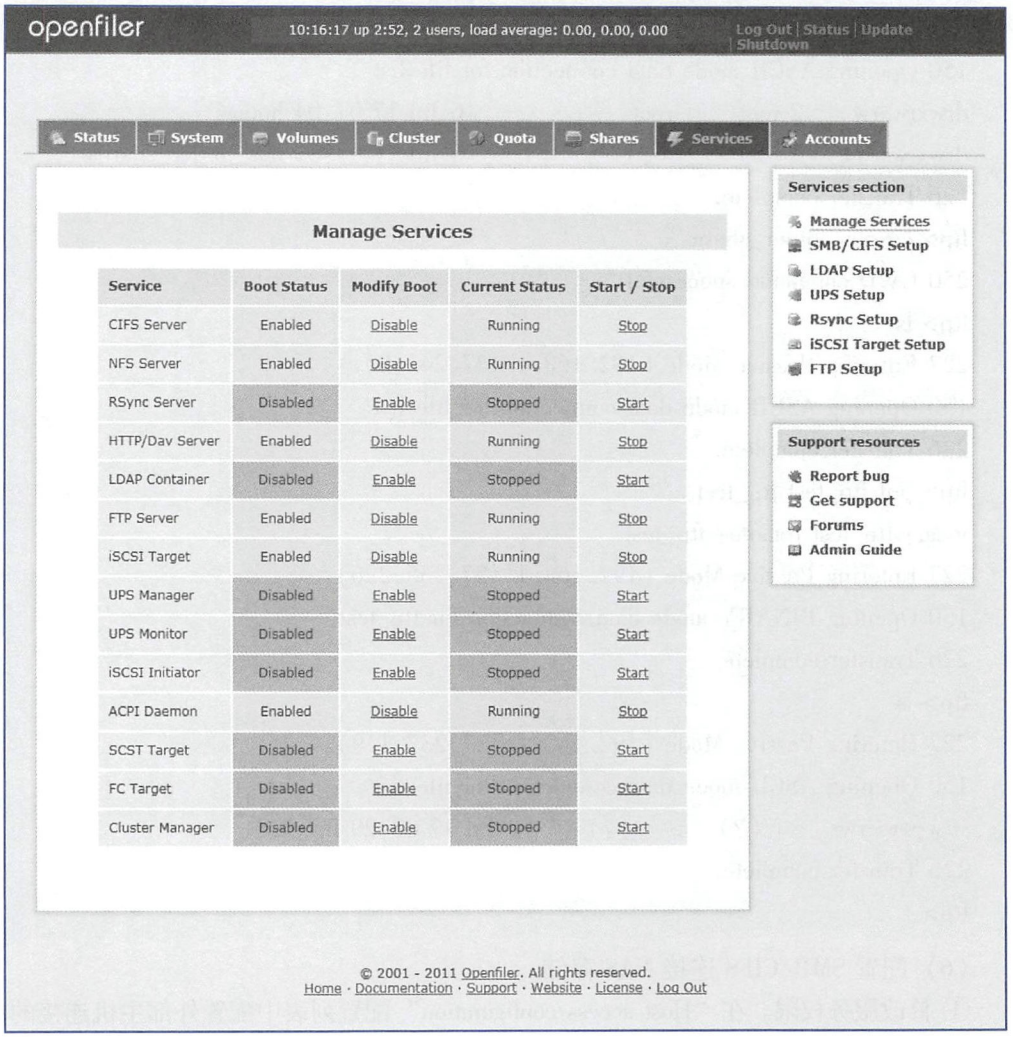


图 2-42 开启 CIFS Server 服务

③ Windows 方式验证，查看 SMB 中的共享文件夹。打开 Windows 资源管理器，在地址栏输入“\\192. 168. 1. 137”，进入服务器，便可以查看创建的 NAS 共享文件夹，如图 2-43 所示。

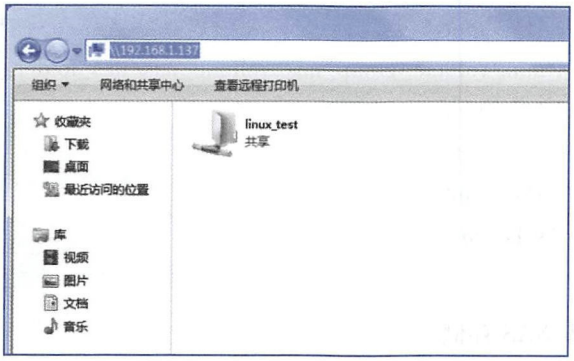


图 2-43　访问 SMB 的共享文件夹

④ 命令行方式验证。查看服务器提供的共享文件夹。

```
[root@ server1 ~]# smbclient -N -L \\192. 168. 1. 137
Domain=[OPENFILER] OS=[Unix] Server=[Samba 3. 5. 6]

        Sharename       Type      Comment
        ---------       ----      -------
        linux_test      Disk      nas_folder_shares
        IPC$            IPC       IPC Service (Openfiler NAS)
Domain=[OPENFILER] OS=[Unix] Server=[Samba 3. 5. 6]

        Server               Comment
        ---------            -------
        Workgroup            Master
        ---------            -------

[root@ server1 ~]# mount -t cifs -o username=openfiler,password=password //
192. 168. 1. 137/linux_test /mnt
[root@ server1 ~]# df -h
Filesystem
Size    Used    Avail    Use%    Mounted on
/dev/mapper/VolGroup-lv_root
6. 7G   1. 1G   5. 3G    17%  /
tmpfs
939M    0       939M     0%      /dev/shm
```

```
    /dev/vda1
485M    32M    428M    7% /boot
    /dev/mapper/myvg-mylv
20G    33M    20G    1% /mnt
    192.168.1.137:/mnt/volume_group_nas/volume_nas/nas_folder_shares
20G    33M    20G    1% /mnt
    //192.168.1.137/linux_test
20G    33M    20G    1% /mnt
    [root@server1 ~]# ls /mnt/
    ftp_test
```

笔记

（7）Rsync 同步 NAS 存储

① 修改服务权限。在“Host access configuration”配置列表中配置外部主机连接的服务权限。将 Rsync 服务设置为 RW，然后单击“Update”按钮生效，如图 2-44 所示。

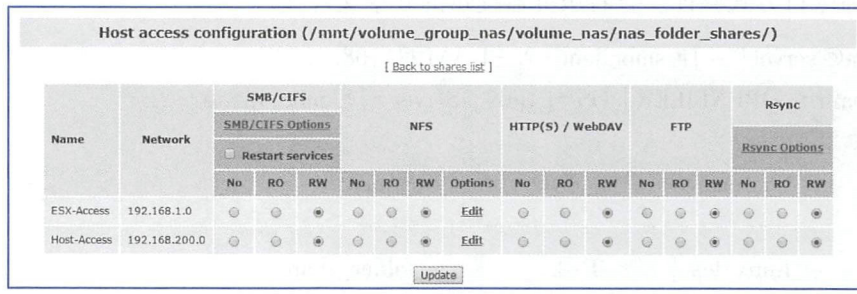


图 2-44　修改 Rsync 服务权限

② 启动 Rsync 服务。选择“Services”选项卡，在界面右侧“Services section”列表中选择“Manage Services”选项。然后将“Manage Services”列表中的 Rsync Server 的 Boot Status 设置为 Enabled 状态，Current Status 设置为 Running 状态，如图 2-45 所示。

③ 同步 NAS 存储文件。将 NAS 共享存储中的文件同步到本地。

```
[root@server1 ~]# rsync -avzP openfiler@192.168.1.137::linux_test /root
Welcome to the Openfiler rsync server.
receiving incremental file list
./
ftp_test
              0 100%    0.00kB/s    0:00:00 (xfer#1, to-check=0/2)

sent 54 bytes  received 152 bytes  19.62 bytes/sec
total size is 0  speedup is 0.00
[root@server1 ~]# ls -l
total 308
```

```
-rw-------. 1 root root    1097 Mar 26   2014 anaconda-ks. cfg
-rw-rw-rw-      1    94    96         0 Jul 17   2017 ftp_test
-rw-r--r--. 1 root root    8815 Mar 26   2014 install. log
-rw-r--r--. 1 root root    3384 Mar 26   2014 install. log. syslog
-rw-r--r--          1 root root   76632 Jul 12 22:23 perl-Config-General-2. 61-
1. el7. noarch. rpm
-rw-r--r--   1 root root 213616 Jul 12 22:23 scsi-target-utils-1. 0. 55-4. el7. x86
_64. rpm
```

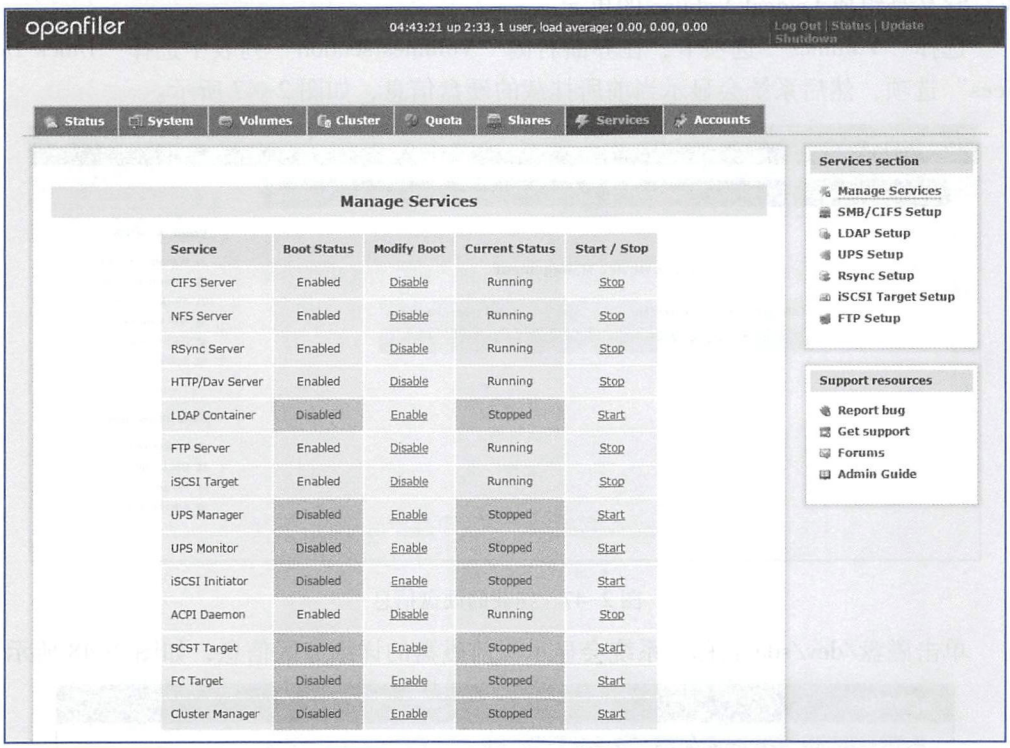


图 2-45　开启 Rsync 服务

（8）搭建 iSCSI 网络存储

① 添加访问控制列表。选择“System”选项卡，在“Network Access Configuration”列表中配置允许访问 Openfiler 系统的安全访问控制列表。值得注意的是只有加入到 Openfiler 网络访问控制列表中的网络或者主机地址，才允许访问 Openfiler 系统所提供的虚拟存储服务。在网络访问控制列表中，既可以配置网段地址也可以配置单个主机地址。本任务在“Network/Host”文本框中输入“192. 168. 1. 0”网段，然后单击“Type”下的下拉按钮▼，在弹出的下拉列表中选择“Share”选项，添加完成后单击“Update”按钮完成系统配置更新，如图 2-46 所示。

② 添加 iSCSI 卷。接下来配置系统磁盘，实现虚拟存储服务。在 Openfiler 中，有 Block Device、Physical Volume 和 Volume Group 3 类存储概念。其中，Block Device 表

笔记

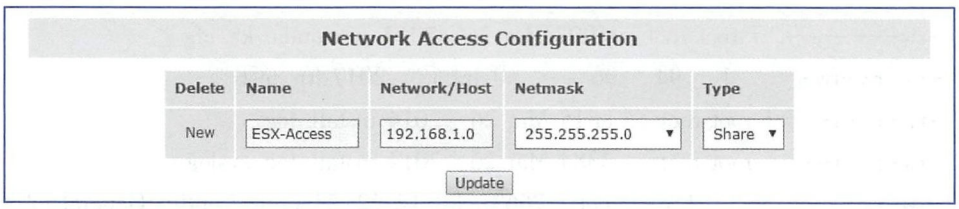


图 2-46 添加访问控制列表

示的是实际的物理磁盘；Physical Volume 表示的是物理磁盘的分区，它是组成 Volume Group 的单元；Volume Group 则是由一个或多个物理磁盘分区（Physical Volume）组成，它又是组成 Logical Volume 的单元。

选择“Volumes”选项卡，在界面右侧“Volumes section”列表中选择“Block Devices”选项，然后系统会显示当前所挂载的硬盘信息，如图 2-47 所示。

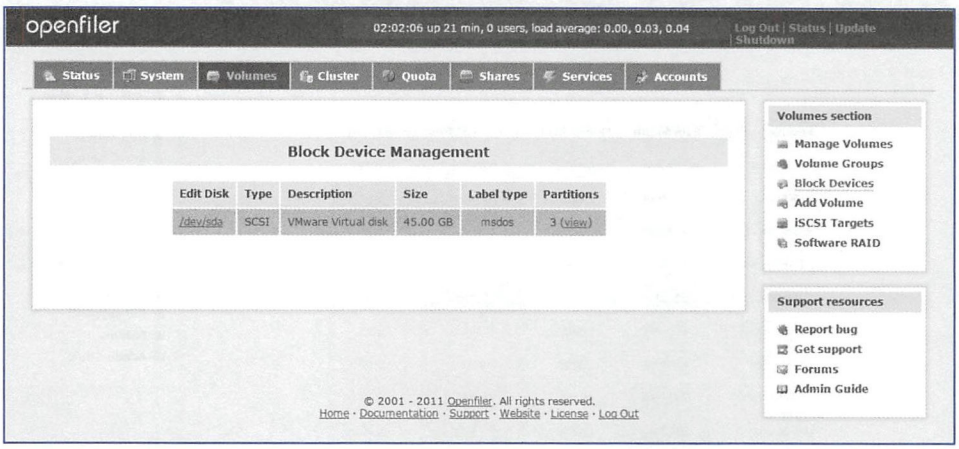


图 2-47 挂载的硬盘信息

单击磁盘/dev/sda 名称，系统会显示当前磁盘的详细分区信息，如图 2-48 所示。

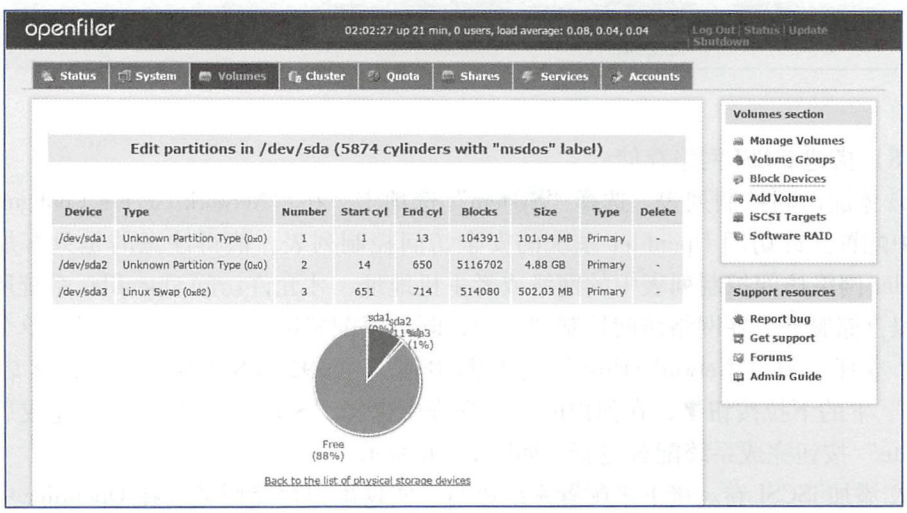


图 2-48 磁盘详细分区信息

在这里，要创建一个新的分区。首先在“Create a partition in/dev/sde”列表下，单击“Partition Type”下的下拉按钮▼，在弹出的下拉列表中选择“Physical volume”选项，然后在“Ending cylinder”属性处设置物理卷的大小，选择物理卷的大小时需要保证比可用卷的空间要小，最后单击“Create”按钮创建，如图 2-49 所示。

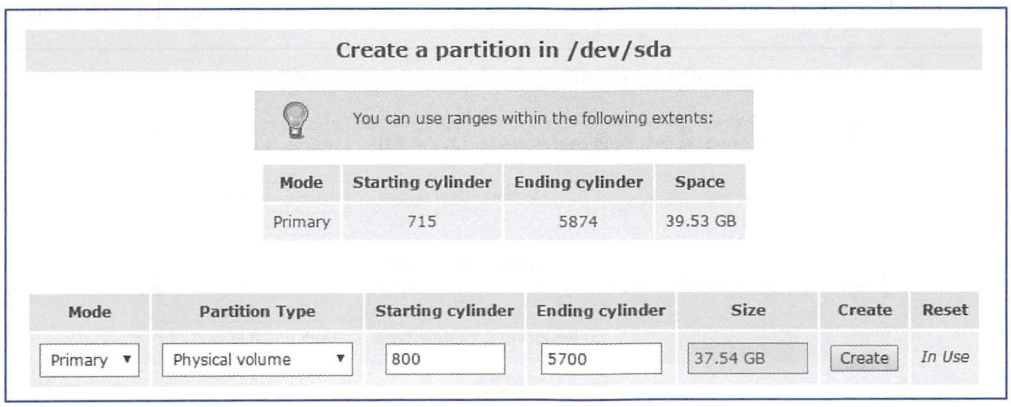


图 2-49　创建新的分区

在界面右侧“Volumes section”列表中选择“Volume Groups”选项，在“Create a new volume group”列表中创建一个卷组。在“volume group name”文本框中输入卷组名称“volume_group_iscsi”，同时选中之前所创建的物理卷/dev/sda4，最后单击“Add volume group”按钮创建，如图 2-50 所示。

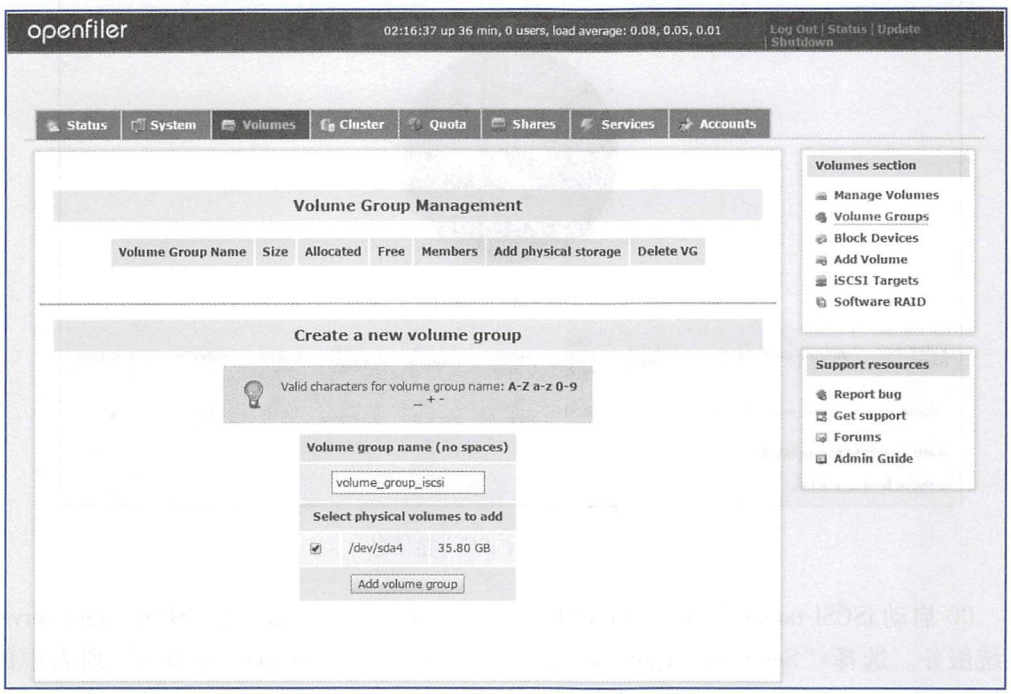


图 2-50　创建卷组

在界面右侧“Volumes section”列表中选择“Add Volume”选项，在刚刚创建的卷组 volume_group_iscsi 中新建一个 iSCSI 卷。在“Volume Name”文本框中输入“volume_iscsi”，在“Volume Description”文本框中输入“volume_supporting_ iscsi”，在“Required Space”文本框中输入“36640”，在“Filesystem / Volume type”下拉菜单中选择“Block（iSCSI，FC，etc）”选项，然后单击“Create”按钮，如图 2-51 所示。

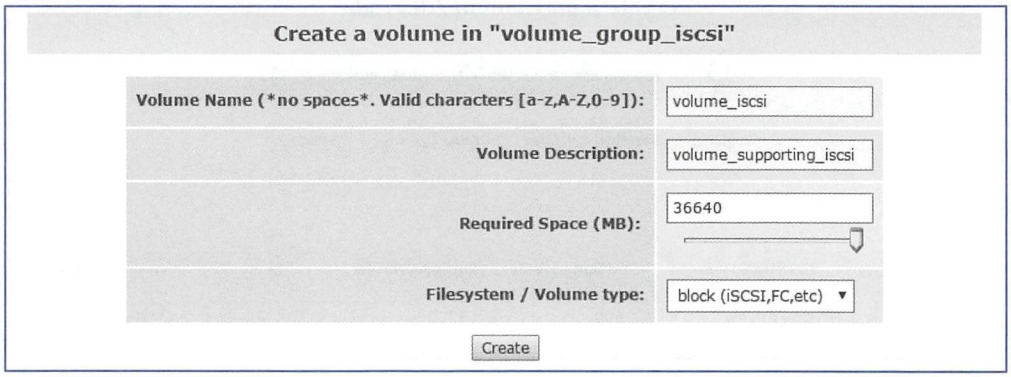


图 2-51 创建 iSCSI 卷

在界面右侧“Volumes section”列表中选择“Manage Volumes”选项，可以查看到刚刚创建的 iSCSI 卷信息，如图 2-52 所示。

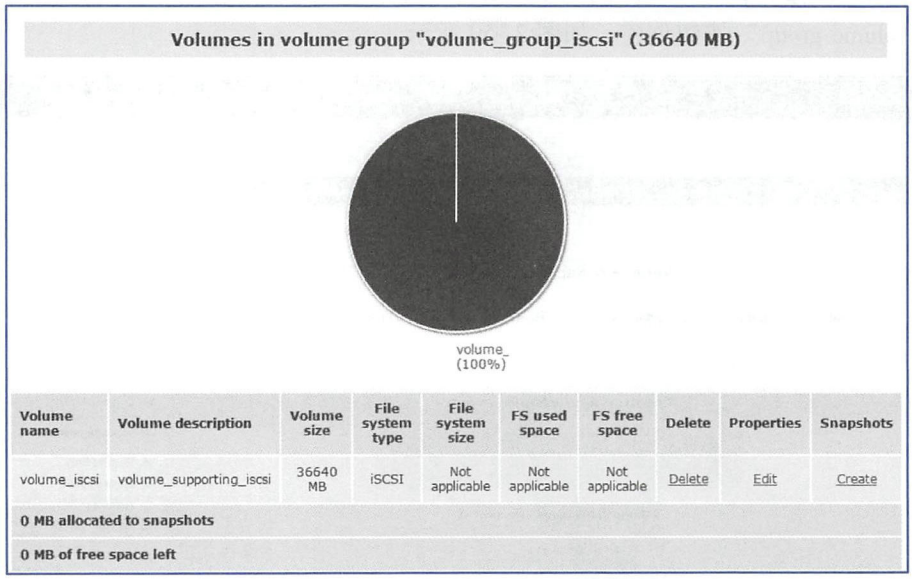


图 2-52 查看 iSCSI 卷信息

③ 启动 iSCSI target 服务。当 iSCSI 卷创建完成后，接下来开启 iSCSI target server 系统服务。选择“Services”选项卡，在界面右侧选择“Services section”列表中的“Manage Services”选项。然后将系统服务列表中的 iSCSI target server 的 Boot Status 设

置为 Enabled 状态，Current Status 设置为 Running 状态，从而使得系统能够对外提供基于 iSCSI 协议的虚拟存储服务，如图 2-53 所示。

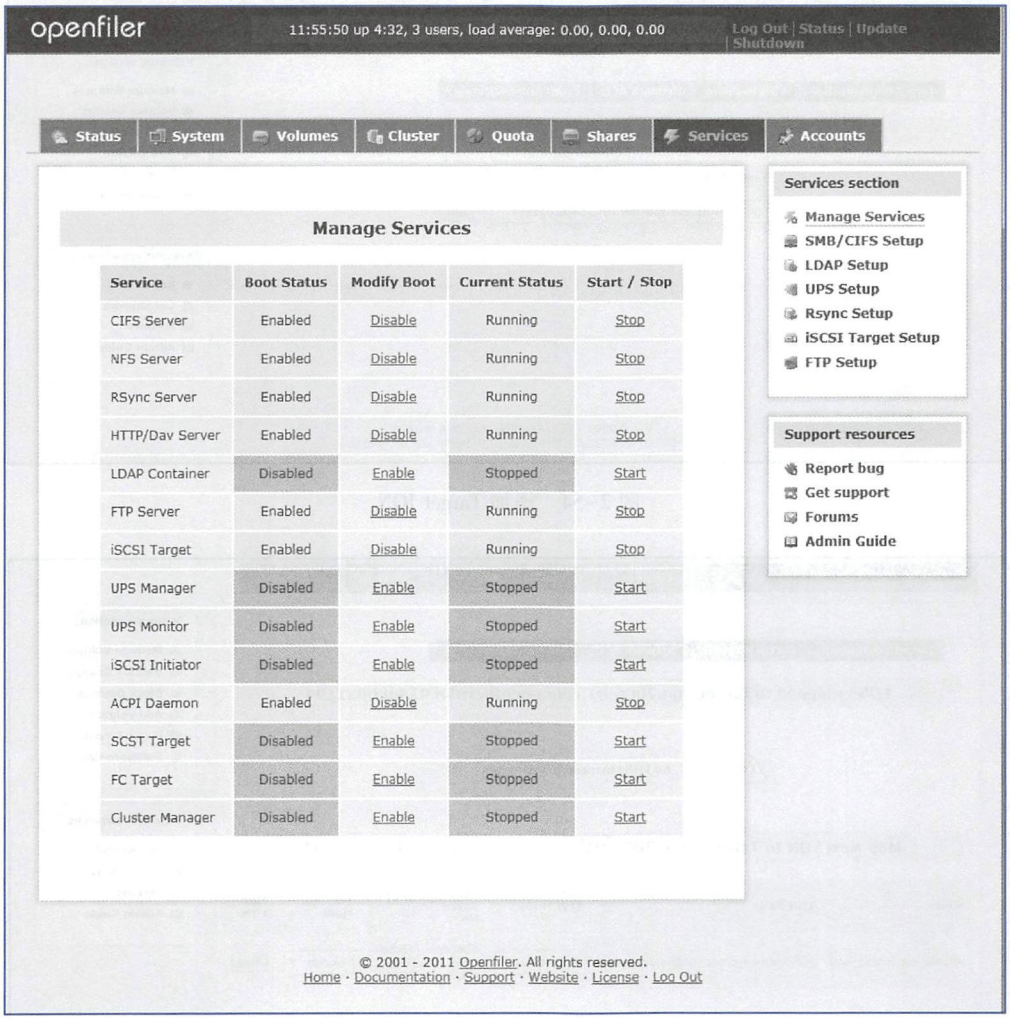


图 2-53　启动 iSCSI Target 服务

笔记

④ LUN 映射。接下来添加一个 iSCSI Target。选择“Volumes”选项卡，在界面右侧“Volumes section”列表中选择“iSCSI Targets”选项。在“Target Configuration”选项卡中单击“Add”按钮，从而添加了一个 iSCSI Target。此处 Target IQN 信息在后面对 ESXi 服务器的存储配置过程中会用到，如图 2-54 所示。

选择“LUN Mapping”选项卡，保持其余选项默认，单击“Map”按钮，从而实现从 LUN 到配置好的 iSCSI Target 之间的映射，如图 2-55 所示。

选择“Network ACL”选项卡，然后单击“Access”下的下拉按钮▼，在弹出的下拉列表中选择“Allow”选项，将访问控制从默认禁止访问更新为允许访问，最后单击“Update”按钮更新系统配置，如图 2-56 所示。

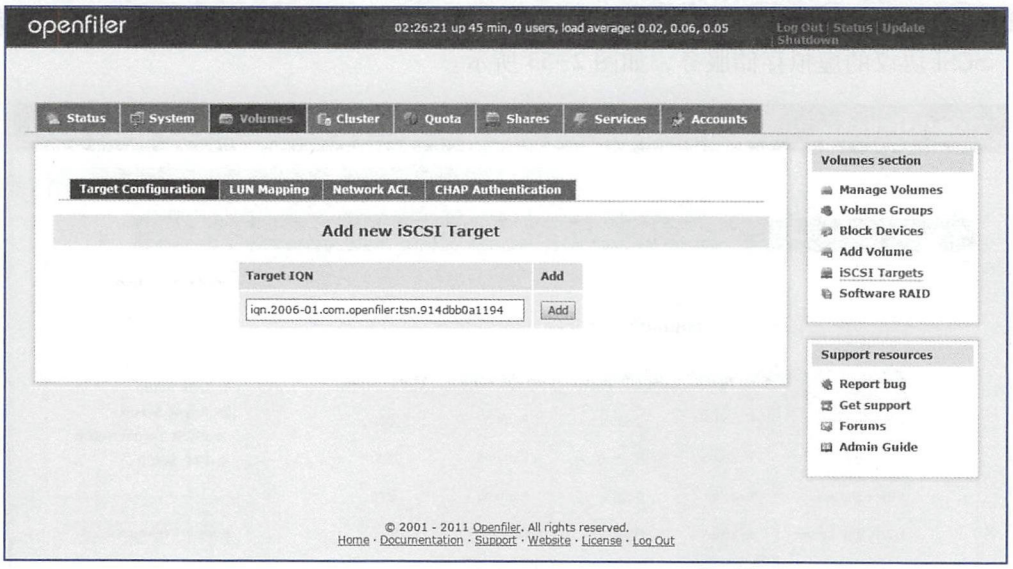


图 2-54 添加 Target IQN

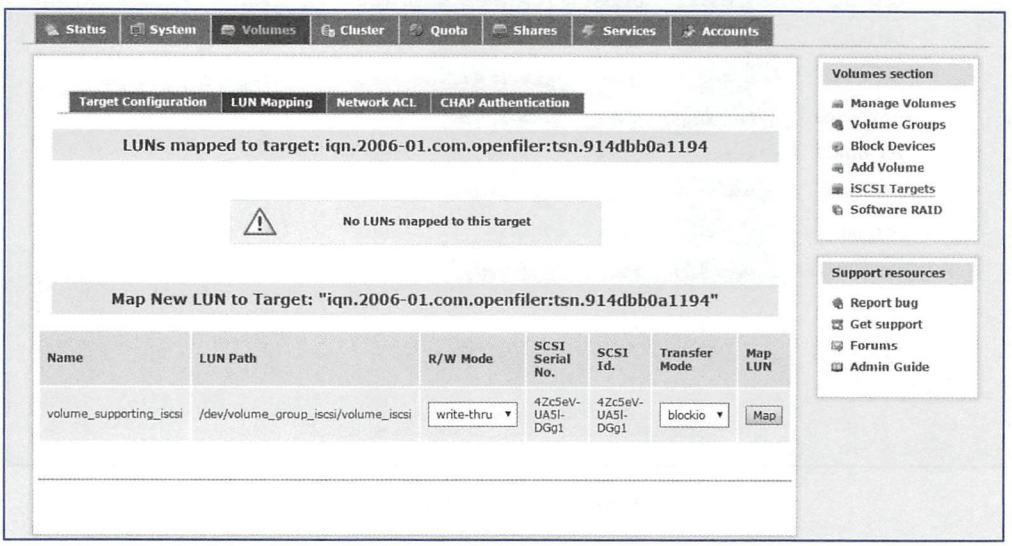


图 2-55 创建 LUN 映射

到此为止，在 Openfiler 系统上对 iSCSI 虚拟存储的配置完毕。

（9）EXSi 连接 iSCSI 网络存储

云计算离不开存储，存储是云计算的基础架构和核心组成部分。在本小节中，将通过一个云端存储配置实例—在云端的虚拟化监控器（Hypervisor）上配置 Openfiler 提供的 iSCSI 虚拟网络磁盘服务，介绍如何利用 Openfiler 这个强大的存储服务软件实现云端的共享存储功能。

① 添加 VMkernel 网络。本示例基于 VMWare ESXi5 虚拟化监控器版本进行配置。

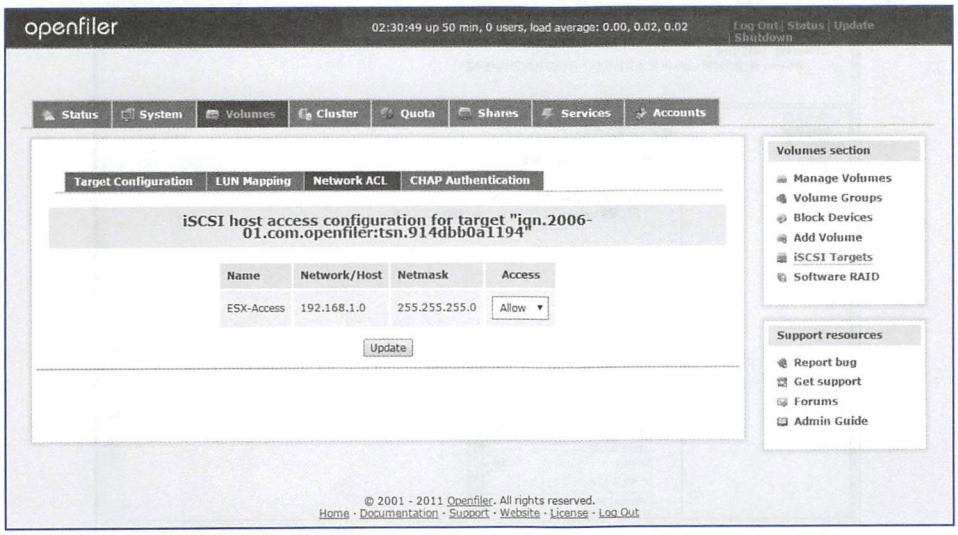


图 2-56　设置访问控制

首先通过客户端软件 VMWare vSphere Client 连接到 ESXi 服务器，然后在主界面左侧列表中选中 ESXi 服务器，接着在主界面右侧选择“配置”选项。在当前界面中选择“硬件”列表的“网络”选项，单击“添加网络”按钮，在打开的“添加网络向导”对话框中选中“VMkernel”单选按钮，然后单击“下一步”按钮，如图 2-57 所示。

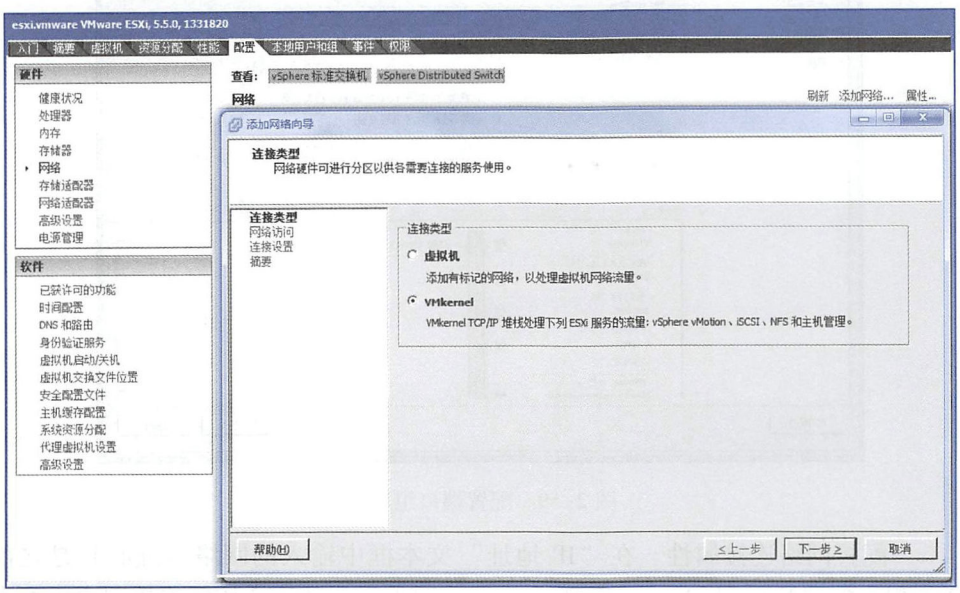


图 2-57　添加网络

在配置本实例中，此次存储网络配置基于当前 ESXi 服务器上已有的交换机 vSwitch0，如图 2-58 所示。

配置网络端口组属性。在“网络标签”文本框中输入“VMKernel”，其余选项保持默认，然后单击“下一步”按钮，如图 2-59 所示。

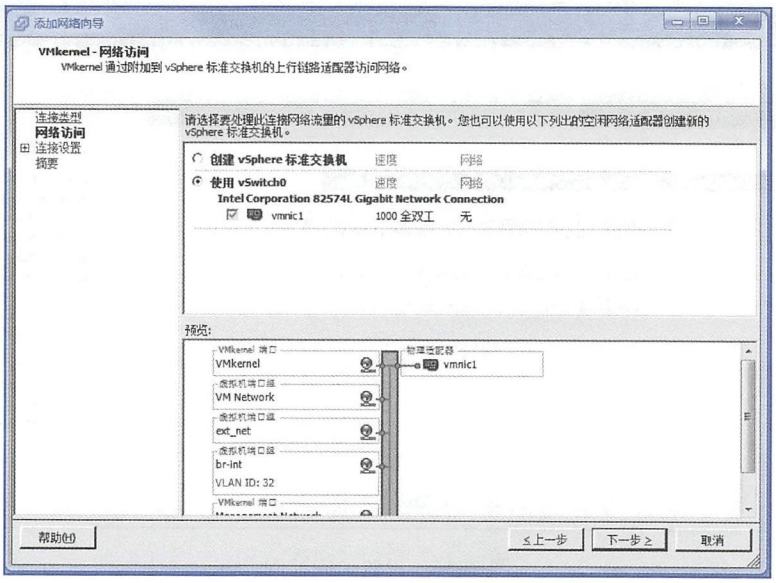


图 2-58 配置网络访问

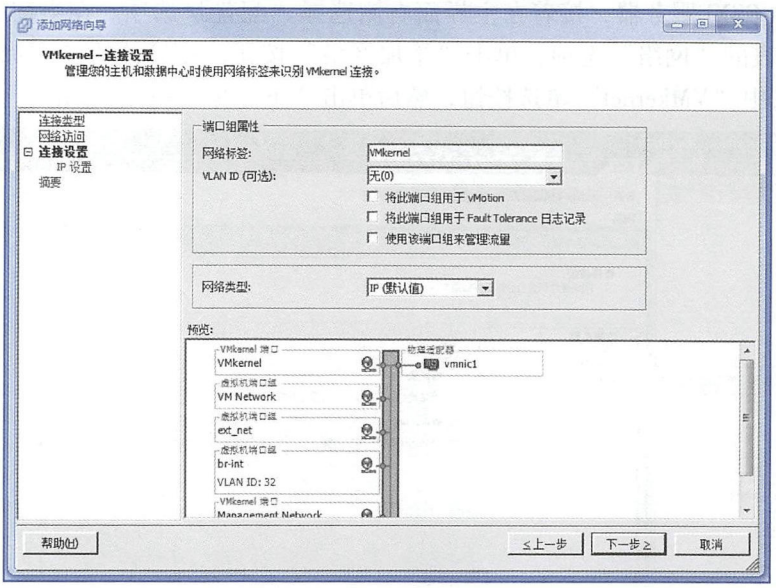


图 2-59 配置端口组属性

笔记

配置网络连接属性。在“IP 地址”文本框中输入的网络地址必须是之前配置 Openfiler 的“Network Access Configuration”时所设置的安全访问连接列表中所允许的网络地址，否则 ESXi 服务器无法访问 Openfiler 所提供的 iSCSI 虚拟网络磁盘服务。填写完后单击“下一步”按钮，如图 2-60 所示。

最后检查此次网络配置的摘要信息，单击“完成”按钮，完成此次网络配置，如图 2-61 所示。

② 配置 iSCSI 存储适配器。配置 ESXi 服务器的 iSCSI 存储适配器。首先在当前界面

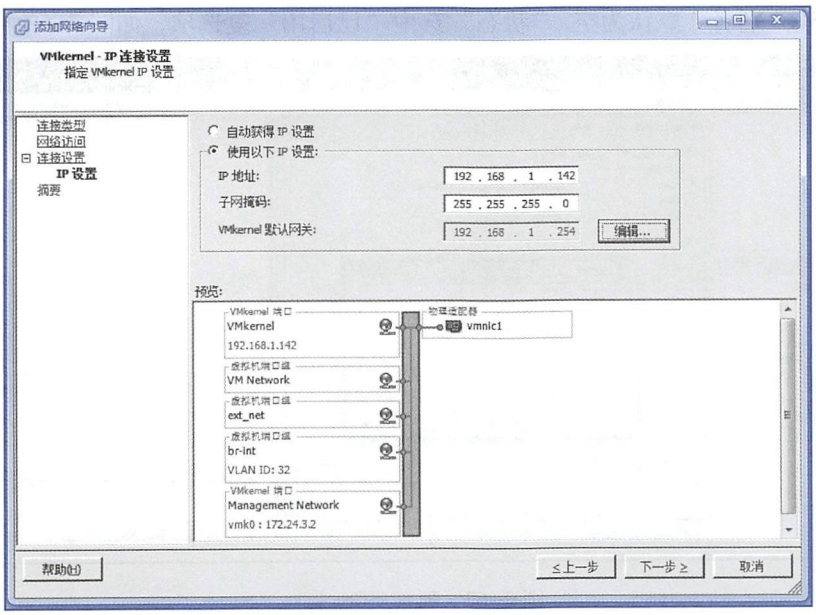


图 2-60　配置网络连接属性

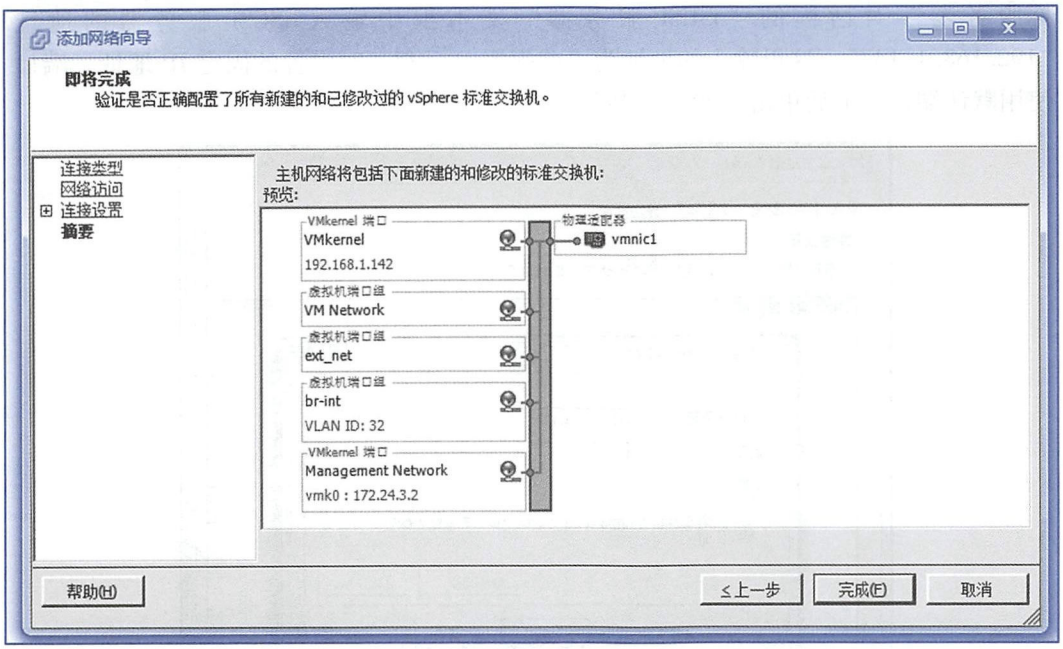


图 2-61　查看摘要信息

选择“配置”选项卡，然后在“硬件”列表中选择“存储适配器”选项。接着在“存储适配器”列表中选择“iSCSI Software Adapter”（如果没有则手动添加），在列表右下方单击“属性”按钮。在打开的“iSCSI 启动器属性”对话框中选择“常规”选项卡，单击“配置”按钮，在打开的“常规属性”对话框中输入 iSCSI 设备的名称。需要注意的是，在这里填写的 iSCSI 设备名称是在之前配置 Openfiler 时所添加的 iSCSI Target IQN

信息，“iSCSI 别名”默认为空，“状态”选中“已启用”复选项，如图 2-62 所示。

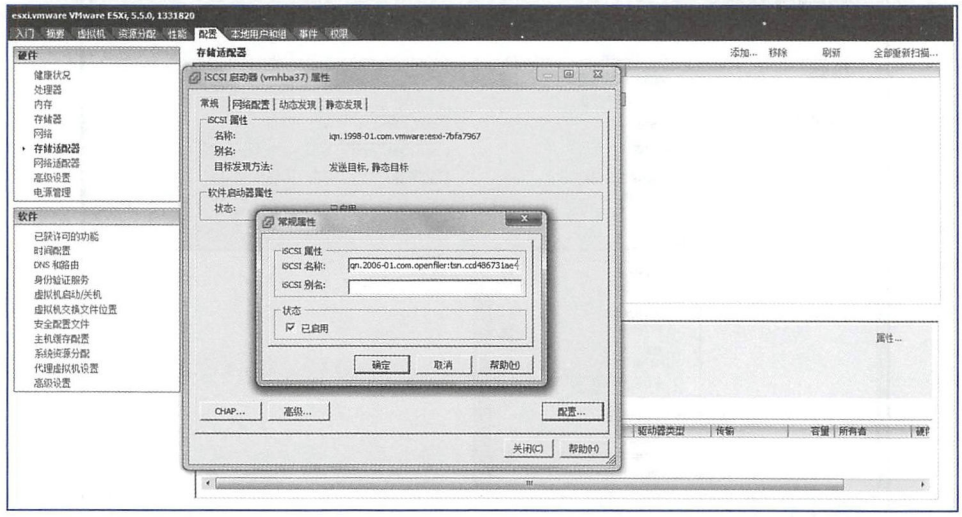


图 2-62 配置 iSCSI 属性

选择“动态发现”选项卡，然后单击下方的“添加”按钮，在打开的“添加目标服务器”对话框的“iSCSI 服务器”文本框中输入 iSCSI 服务器的地址“192.168.1.137”。这里输入的地址为之前配置 Openfiler 时设置的固定 IP 地址。端口使用默认端口，最后单击“确定”按钮，如图 2-63 所示。

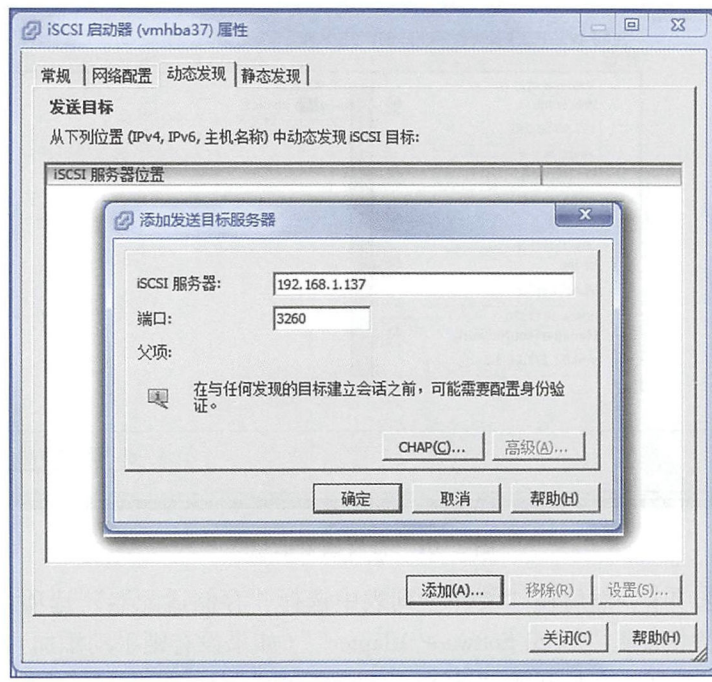


图 2-63 配置 iSCSI 服务器 IP 地址

系统会提示 ESXi 服务器将要基于这次配置变化对主机总线适配器进行重新扫描，在打开的“重新扫描”对话框中单击“是”按钮，如图 2-64 所示。

笔记

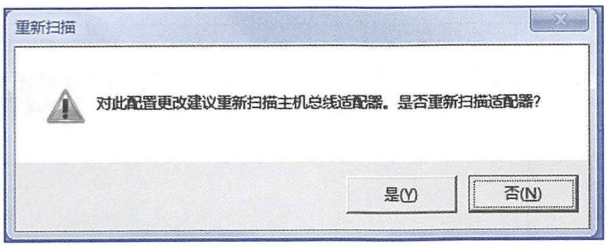


图 2-64　重新扫描存储适配器提示

③ 添加 iSCSI 存储。对 ESXi 服务器的 iSCSI 适配器配置完成后，接下来是为 ESXi 服务器添加相应的 iSCSI 存储。首先选择“配置”选项卡，然后在界面左侧“硬件”列表中选择“存储器”选项，在界面右侧单击“添加存储器”按钮。在打开的“添加存储器”对话框中选中“存储器类型”下的“磁盘/LUN”单选按钮，然后单击“下一步”按钮，如图 2-65 所示。

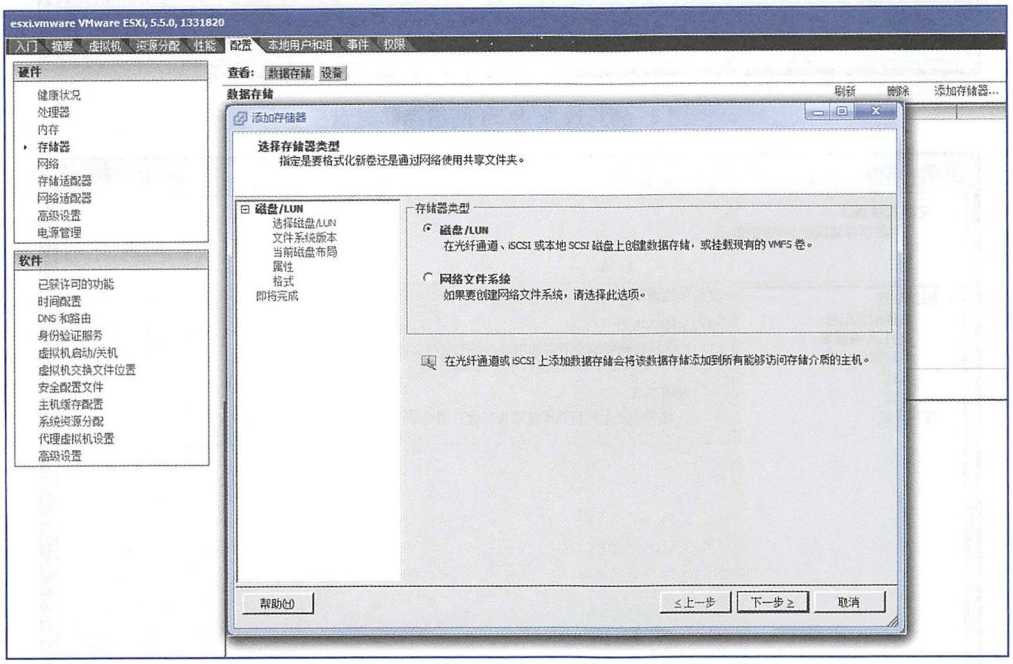


图 2-65　选择存储器类型

选择刚刚配置的 iSCSI 网络存储，单击“下一步”按钮，如图 2-66 所示。

选中“文件系统版本”的“VMFS-5”单选按钮，然后单击“下一步”按钮，如图 2-67 所示。

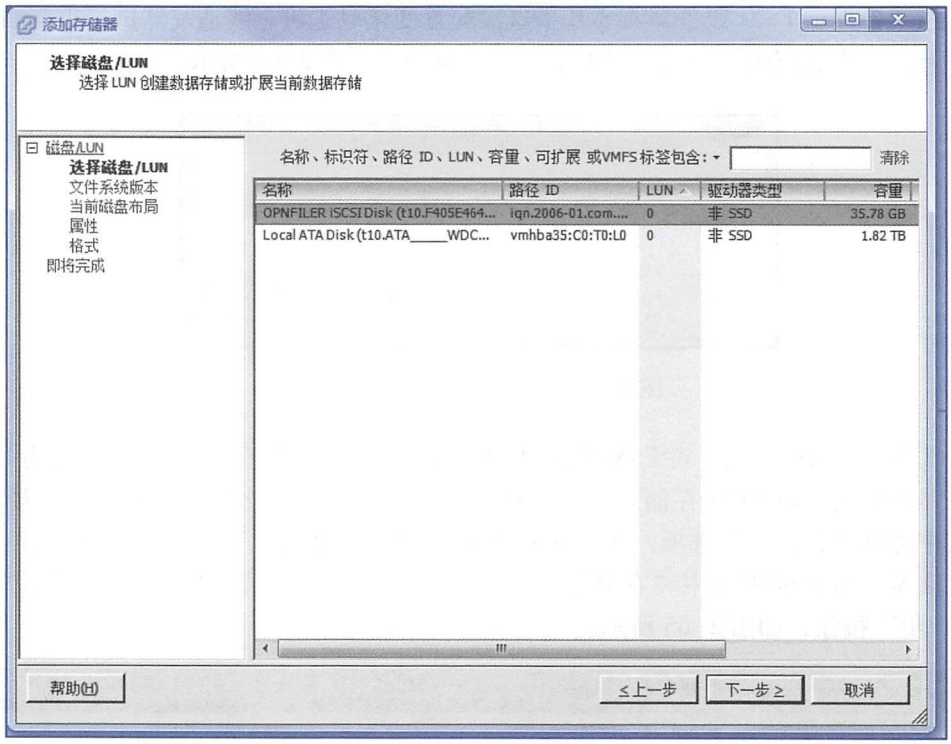


图 2-66 选择 iSCSI 网络存储磁盘

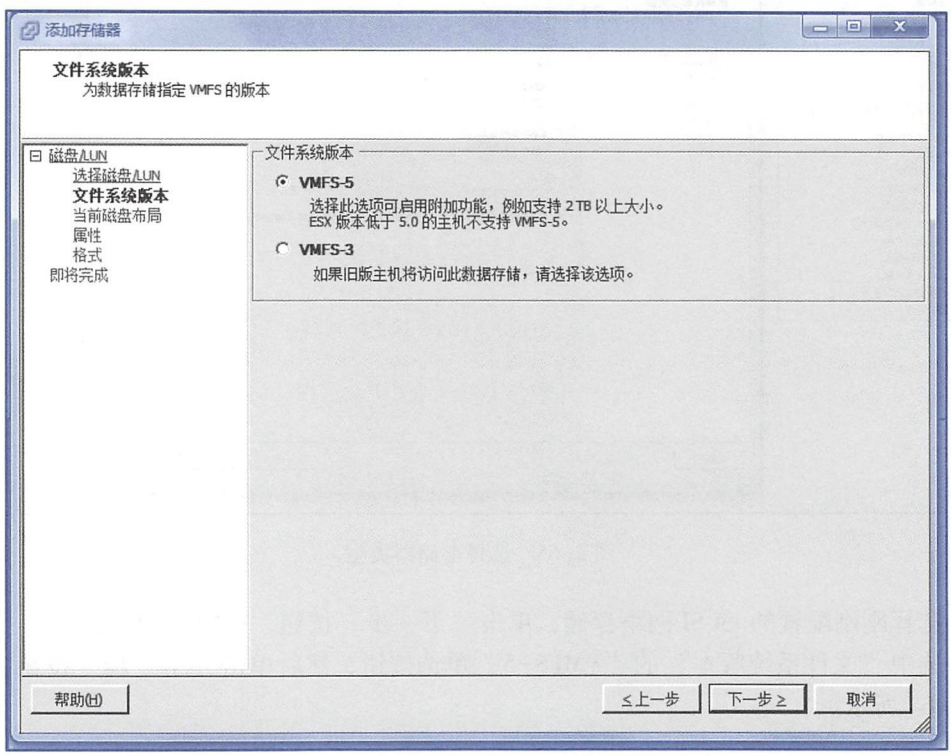


图 2-67 选择文件系统版本

此时 iSCSI 磁盘处于初始化空白状态，单击“下一步”按钮，如图 2-68 所示。

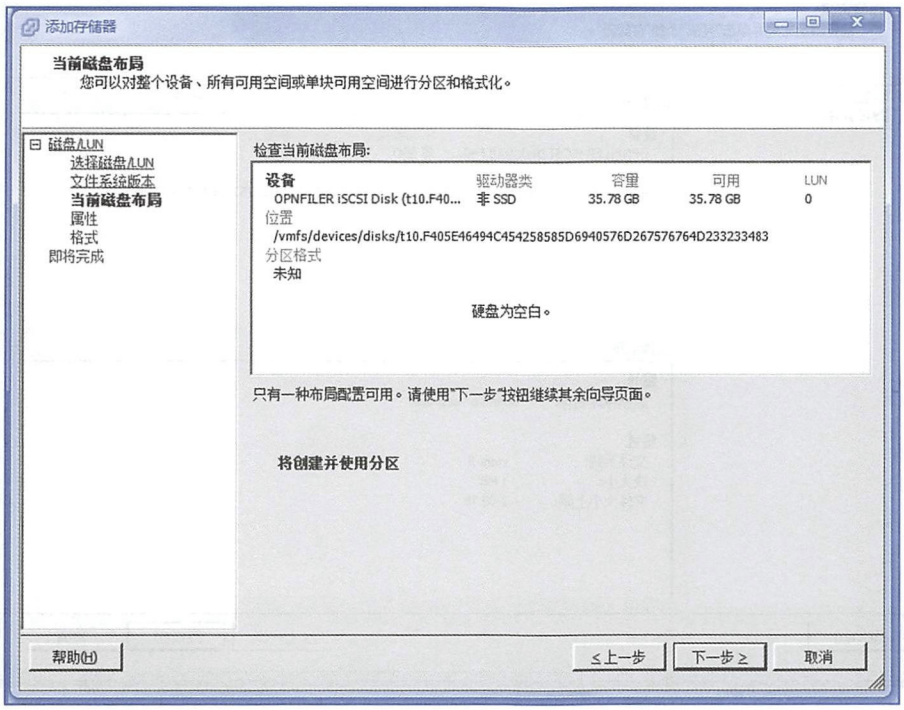


图 2-68　检查当前磁盘布局

设置数据存储的名称，单击“下一步”按钮。接下来用户可以根据不同应用环境的需要调整新建存储的容量。本任务选中“最大可用空间”单选按钮，即选择默认格式化大小容量设置，单击“下一步”按钮，如图 2-69 所示。最后确认新建存储配置信息无误后，单击“完成”按钮，新建存储便设置完成了，如图 2-70 所示。

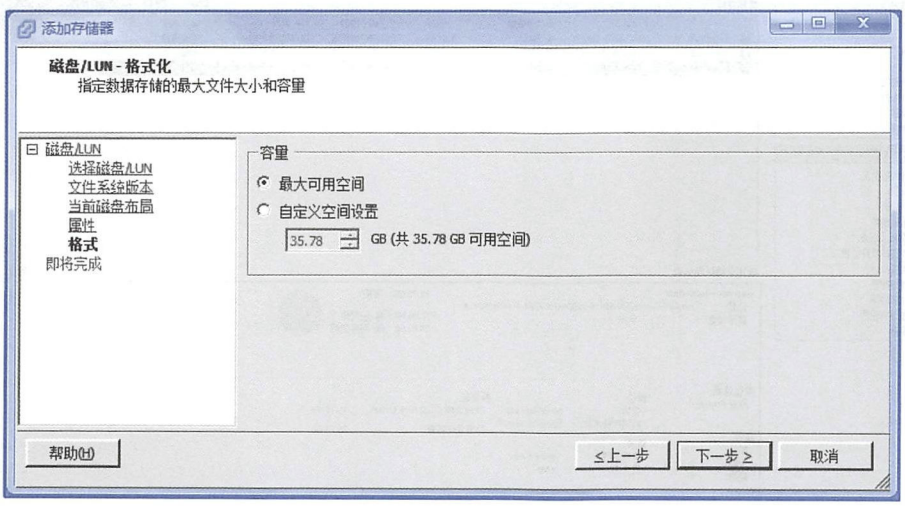


图 2-69　调整存储可用空间

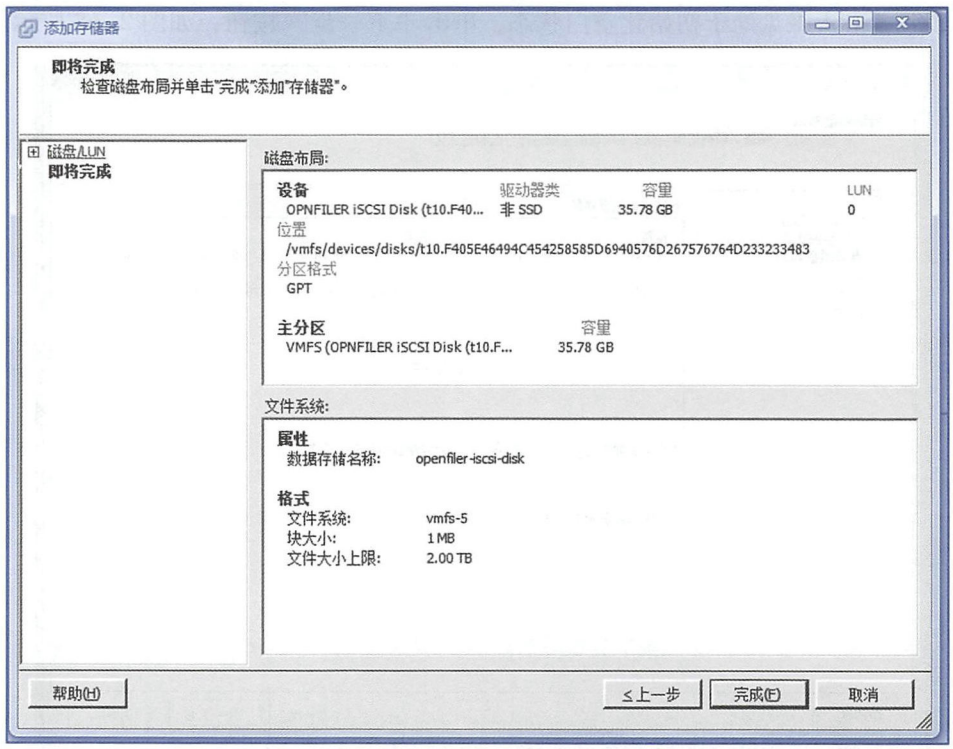


图 2-70 确认新建存储配置信息

当 ESXi 服务器对新建的 iSCSI 数据存储服务配置完成后，可以看到新建 iSCSI 存储的详细信息，如图 2-71 所示。

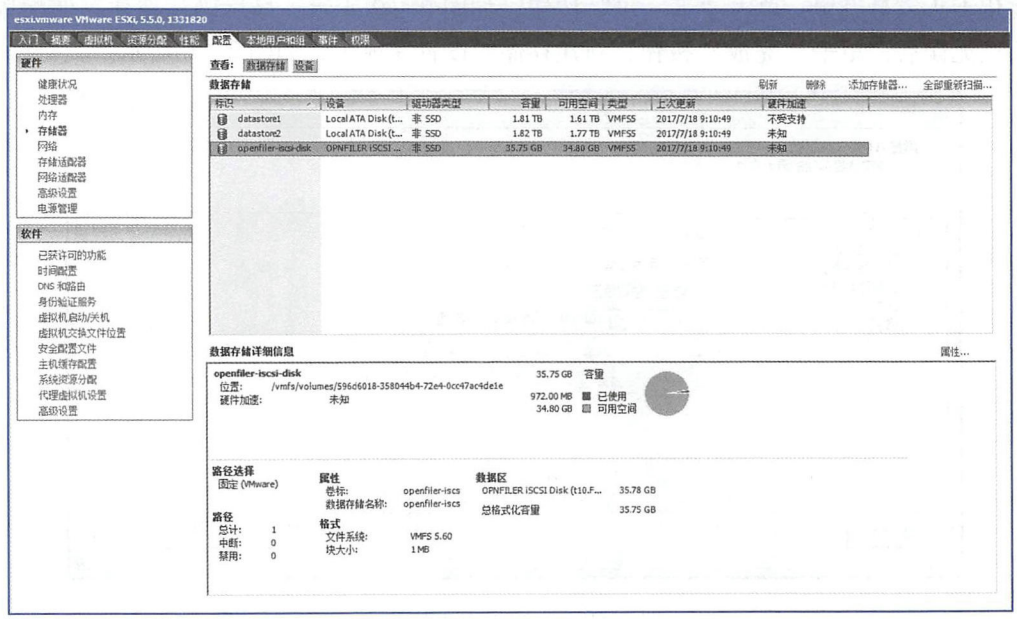


图 2-71 新建 iSCSI 存储详细信息

④ 验证 iSCSI 存储。最后，为了验证在 ESXi 服务器上配置的基于 Openfiler 的 iSCSI 存储服务是否成功，在此通过从本地上传一个测试文件进行验证，如图 2-72 所示。

笔记

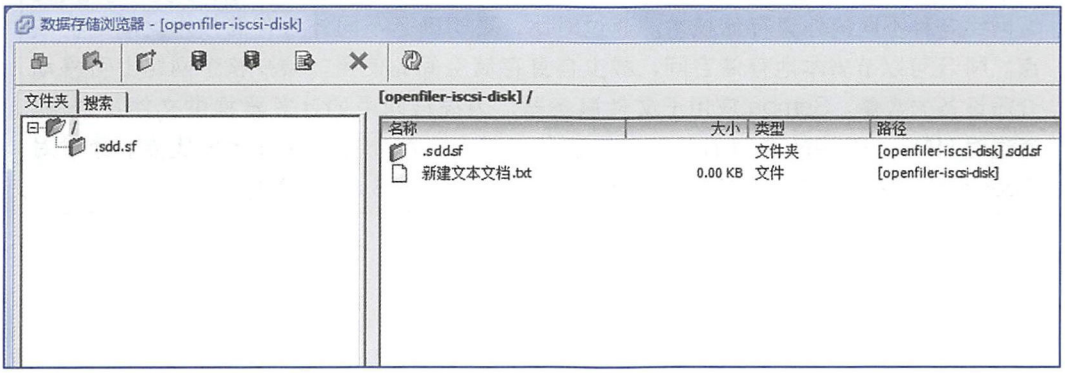


图 2-72　上传文件至 iSCSI 存储

从图 2-72 可以看到，测试文件能够成功上传并存储在新建的 iSCSI 存储服务上。至此，在 ESXi 服务器上成功地配置了基于 Openfiler 的 iSCSI 存储服务，接下来可以以此 iSCSI 存储服务在 ESXi 服务器上进行云端虚拟机的创建了。

项目实训

【实训题目】

使用 Openfiler 构建 NAS 和 IP SAN。

【实训目的】

1. 掌握 ESXi 操作系统的安装。
2. 掌握 Openfiler 操作系统的安装，使用 Web 方式管理 Openfiler。
3. 掌握 NAS 存储网络的创建。
4. 掌握 iSCSI 存储网络的创建。

【实训内容】

1. 安装 ESXi 操作系统。配置 ESXi 系统相关信息，Root 密码设置为 PassWord，并配置 IP 地址。

2. 安装 Openfiler 操作系统，上传镜像包并安装虚拟机，进入 Openfiler 管理 Web 页面。

3. 创建 NAS 存储网络、创建 NAS 卷、配置 NAS 存储服务、开启共享服务、设置 FTP 连接 NAS 存储、添加访问控制列表和访问主机所在网段。

4. 搭建 iSCSI 网络存储，添加访问控制列表和 iSCSI 卷并启动 iSCSI Target 服务。

5. 利用 EXSI 连接 iSCSI 网络存储，并添加 iSCSI 存储。通过上传文件确定是否添加成功。

单元小结

本单元介绍了外置存储技术的 DAS 直连式存储、NAS 网络接入存储、SAN 存储区域网络 3 种不同的外置存储技术。通过对比，更明确了不同外置存储技术之间的优缺点。NFS 可以节省本地存储空间，减少自身存储空间的使用，减少整个网络上可移动介质设备的数量。Samba 应用于文件服务器，为局域网中的计算器提供文件及资源。通过对 NAS、IP SAN 环境的部署，使读者更深入了解外置存储技术的优点。通过对 Openfiler、NAS、iSCSI 的学习，使用 Openfiler 实现 NAS 和 IPSAN。

单元 3

构建OpenStack云存储系统

学习目标

【知识目标】

- 了解对象存储、块存储的概念。
- 了解对象存储、块存储的使用场景。
- 了解私有云的概念和搭建。

【技能目标】

- 掌握 OpenStack 私有云的搭建。
- 掌握块存储在私有云中的使用。
- 掌握对象存储在私有云中的使用。

PPT3：构建 Open Stack 云存储系统

学习情境

小缪在之前的研究过程中对服务器的基本存储技术都已经熟练掌握了，觉得公司可以在内部搭建一个私有云平台来满足公司日益增长的需求，将数据统一存储在云端，于是研发部让小缪研究私有云的存储方案。

（1）项目设计

搭建单节点的私有云平台。

（2）存储基本规划

① 实现 OpenStack Cinder 块存储。

② 实现 OpenStack Swift 对象存储。

（3）服务器功能实现

搭建完成私有云，可以正常创建云主机并使用，Cinder 和 Swift 存储也可以正常使用。

任务 3.1 构建 OpenStack 私有云

任务描述

1. 了解 OpenStack 的服务组成。
2. 掌握 OpenStack 的搭建方法和 OpenStack 的使用方法。

知识学习

笔记

1. 了解 OpenStack

OpenStack 是一个由 NASA（美国国家航空航天局）和 Rackspace（全球三大云计算中心之一）合作研发并发起的，阿帕奇（Apache HTTP Server，简称 Apache，世界使用排名第一的 Web 服务器软件）许可授权的自由软件和开放源代码项目。

OpenStack 是一个开源的云计算管理平台项目，由几个主要的组件组合起来完成具体工作。OpenStack 支持几乎所有类型的云环境，项目目标是提供实施简单、可大规模扩展、丰富、标准统一的云计算管理平台。OpenStack 通过各种互补的服务提供了基础设施即服务（IaaS）的解决方案，每个服务提供 API 以进行集成。

OpenStack 是一个旨在为公共及私有云的建设与管理提供软件的开源项目。它的社区拥有超过 130 家企业及 1350 位开发者，这些机构与个人都将 OpenStack 作为基础设施即服务（IaaS）资源的通用前端。OpenStack 项目的首要任务是简化云的部署过程并为其带来良好的可扩展性。

OpenStack 帮助服务商和企业内部实现类似于 Amazon EC2（Elastic Compute Cloud，亚马逊弹性计算云）和 S3（Simple Storage Service，存储平台）的云基础架构服务（Infrastructure as a Service，IaaS）。OpenStack 包含两个主要模块 Nova 和 Swift，前者是 NASA 开发的虚拟服务器部署和业务计算模块，后者是 Rackspace 开发的分布式云存储模块，两者可以一起使用，也可以分开单独使用。OpenStack 除了有 Rackspace 和 NASA 的大力支持外，还有包括戴尔（Dell）、思杰（Citrix）、思科（Cisco）、Canonical 等重量级公司的贡献和支持，发展速度非常快，有取代另一个业界领先开源云平台 Eucalyptus 的态势。

2. OpenStack 核心服务

OpenStack 覆盖了网络、虚拟化、操作系统、服务器等各个方面。它是一个正在开发中的云计算平台项目，根据成熟及重要程度的不同，被分解成核心项目、孵化项目以及支持项目和相关项目。每个项目都有自己的委员会和项目技术主管，而且每个项目都不是一成不变的。如孵化项目可以根据发展的成熟度和重要性，转变为核心项目。截止到 OpenStack 的 Icehouse 版本，以下列出了几个核心项目（即 OpenStack 核心服务）。

① Keystone 身份服务（Identity Service），为 OpenStack 其他服务提供身份验证、服务规则和服务令牌的功能，管理 Domains（域）、Projects（项目）、Users（用户）、Groups（群组）、Roles（角色）。自 Essex 版本集成到项目中。

② Glance 镜像服务（Image Service），是一套虚拟机镜像查找及检索系统，支持多种虚拟机镜像格式（AKI、AMI、ARI、ISO、QCOW2、Raw、VDI、VHD、VMDK），有创建上传镜像、删除镜像、编辑镜像基本信息的功能。自 Bexar 版本集成到项目中。

③ Nova 计算服务（Compute），用于为单个用户或使用群组管理虚拟机实例的整个生命周期，根据用户需求来提供虚拟服务。负责虚拟机创建、开机、关机、挂起、暂停、调整、迁移、重启、销毁等操作，配置 CPU、内存等信息规格。自 Austin 版本集成到项目中。

④ Swift 对象存储（Object Storage），是一套用于在大规模可扩展系统中，通过内置冗余及高容错机制实现对象存储的系统，允许用户进行存储或者检索文件。可为 Glance 提供镜像存储，为 Cinder 块存储提供卷备份服务。自 Austin 版本集成到项目中。

⑤ Neutron 网络管理（Network），提供了云计算的网络虚拟化技术，为 OpenStack 其他服务提供网络连接服务。为用户提供接口，可以定义 Network、Subnet（子网）、Router（路由器），配置 DHCP、DNS、负载均衡、L3 服务，网络支持 GRE、VLAN 等。其插件架构支持许多主流的网络厂家和技术，如 Open-vSwitch（开放虚拟交换标准）。自 Folsom 版本集成到项目中。

笔记

⑥ Cinder 块存储（Block Storage），为运行实例提供稳定的数据块存储服务，它的插件驱动架构有利于块设备的创建和管理，如创建卷、删除卷，在实例上挂载和卸载卷。自 Folsom 版本集成到项目中。

⑦ Horizon 控制台（Dashboard），是 OpenStack 中各种服务的 Web 管理门户，用于简化用户对服务的操作，如启动实例、分配 IP 地址、配置访问控制等。自 Essex 版本集成到项目中。

任务实施

（1）配置安装环境

① 环境：最小化安装的 CentOS 6.5 的服务器，安装 All-in-One 的 Iaas 平台。

② 镜像文件：CentOS-6.5-x86_64-bin_DVD,XianDian-IaaS-v1.4.iso。

（2）修改基本配置

① 修改主机名为 controller。

```
[root@localhost ~]# hostname controller
```

修改/etc/sysconfig/network 文件，把里面的 hostname 改成 controller。

```
[root@localhost ~]# vi /etc/sysconfig/network
NETWORKING=yes
```

```
HOSTNAME=controller
NOZEROCONF=yes
```

② 配置网络。

配置 eth0：192.168.100.10。

```
# vi /etc/sysconfig/network-scripts/ifcfg-eth0
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=static
IPADDR=192.168.100.10
GATEWAY=192.168.100.1
NETMASK=255.255.255.0
```

配置 eth1：192.168.200.10。

```
# vi/etc/sysconfig/network-scripts/ifcfg-eth1
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

③ 配置 YUM 源，使用镜像文件作为本地源。

```
# mount -o loop CentOS-6.5-x86_64-bin_DVD /mnt
# cp -rfv /mnt/* /opt/centos
```

如果/opt/centos 目录不存在的话，新建一个。

```
# umount /mnt
# mount -o loop XianDian-IaaS-v1.4.iso /mnt
# cp -rfv /mnt/* /opt
```

此时的/opt 目录下应该是如下代码。

```
[root@controller ~]# cd /opt/
[root@controller opt]# ll
total 4
drwxr-xr-x. 8 root root 4096 May  2 08:27 centos
drwxr-xr-x. 4 root root   32 May  2 12:12 iaas-repo
drwxr-xr-x. 2 root root   97 May  2 12:12 images
```

配置 yum 的配置文件。

```
# cd /etc/yum.repos.d/
```

删掉原来的所有文件。

```
# rm -rf *
```

创建一个 local. repo，添加如下代码。

```
# vi local. repo
[centos]
name=centos
baseurl=file:///opt/centos
gpgcheck=0
enabled=1
[iaas]
name=iaas
baseurl=file:///opt/iaas-repo
gpgcheck=0
enabled=1
```

保存退出，执行如下命令。

```
# yum clean all
# yum list
```

④ 关闭防火墙，设置防火墙开机不启动。

```
[root@ controller ~]# service iptables stop
[root@ controller ~]# chkconfig iptables off
```

⑤ 关闭 SELinux。

临时关闭。

```
[root@ controller ~]# setenforce 0
```

永久关闭。

```
[root@ controller ~]# vi /etc/selinux/config
把 SELINUX=enforcing 改成 SELINUX=permissive
```

⑥ 安装 iaas-xiandian。

```
[root@ controller ~]# yum install iaas-xiandian -y
```

⑦ 修改全局配置文件 openrc. sh。

```
[root@ controller ~]# vi /etc/xiandian/openrc. sh
##--------------------system config--------------------##
##Controller Server Manager IP. example:x. x. x. x
HOST_IP=192. 168. 100. 10
```

```
##Controller Server hostname. example:controller
HOST_NAME=controller

##Compute Node Manager IP. example:x. x. x. x
HOST_IP_NODE=192. 168. 100. 10

##Compute Node hostname. example:compute
HOST_NAME_NODE=controller

##--------------------MySQL config--------------------##
##Password for MySQL root user . exmaple:000000
DB_PASS=000000

##--------------------Keystone config-----------------##
##Password for Keystore admin user. exmaple:000000
ADMIN_PASS=000000

##Password for Mysql keystore user. exmaple:000000
KEYSTONE_DBPASS=000000

##--------------------Glance config-------------------##
##Password for Mysql glance user. exmaple:000000
GLANCE_DBPASS=000000

##Password for Keystore glance user. exmaple:000000
GLANCE_PASS=000000

##--------------------Nova config---------------------##
##Password for Mysql nova user. exmaple:000000
NOVA_DBPASS=000000

##Password for Keystore nova user. exmaple:000000
NOVA_PASS=000000

##--------------------Neturon config------------------##
##Password for Mysql neutron user. exmaple:000000
```

```
NEUTRON_DBPASS=000000

##Password for Keystore neutron user.  exmaple:000000
NEUTRON_PASS=000000

##metadata secret for neutron.  exmaple:000000
METADATA_SECRET=000000

##External Network Interface.  example:eth1
INTERFACE_NAME=eth1

##First Vlan ID in VLAN RANGE for VLAN Network.  exmaple:101
#minvlan=

##Last Vlan ID in VLAN RANGE for VLAN Network.  example:200
#maxvlan=

##--------------------Cinder config--------------------##
##Password for Mysql cinder user.  exmaple:000000
CINDER_DBPASS=000000

##Password for Keystore cinder user.  exmaple:000000
CINDER_PASS=000000

##Cinder Block Disk. example:md126p3
BLOCK_DISK=sda4

##--------------------Swift config--------------------##
##Password for Keystore swift user.  exmaple:000000
SWIFT_PASS=000000

##The NO1.  NODE Object Disk for Swift.  example: md126p4.  The 2nd will be
OBJECT_DISK_2
OBJECT_DISK_1=sda3

##The NO1.  NODE IP for Swift Storage Network.  example:x. x. x. x.  The 2nd will be
STORAGE_LOCAL_NET_IP_2
```

```
STORAGE_LOCAL_NET_IP_1=192.168.100.10

##The numbers of all the Swift Nodes. example:3
SWIFT_ZONE=1

##The number of this Swift Node. exmaple:1
SWIFT_NODE=1

##--------------------Heat config----------------------##
##Password for Mysql heat user. exmaple:000000
#HEAT_DBPASS=

##Password for Keystore heat user. exmaple:000000
#HEAT_PASS=

##--------------------Ceilometer config----------------##
##Password for Mysql ceilometer user. exmaple:000000
#CEILOMETER_DBPASS=

##Password for Keystore ceilometer user. exmaple:000000
#CEILOMETER_PASS=

##token for ceilometer. exmaple:000000
#CEILOMETER_TOKEN=

##--------------------Sahara config--------------------##
##Password for Mysql sahara user. exmaple:000000
#SAHARA_DBPASS=

##Password for Keystore sahara user. exmaple:000000
#SAHARA_PASS=
```

按照上面的配置文件填写。

任务需要在本机上划两个分区，本任务划分了/dev/sda3 和/dev/sda4 两个分区，一个为 Swift 使用，一个为 Cinder 使用。

⑧ 安装 Qpid 服务。

```
[root@ controller ~]# yum install -y qpid-cpp-server
```

编辑配置文件。

```
[root@ controller ~]# vi /etc/qpidd.conf
```

修改最后一行代码。

```
auth=no
```

启动服务。

```
[root@ controller ~]# service qpidd start
```

设置开机启动。

```
[root@ controller ~]# chkconfig qpidd on
```

⑨ 修改/etc/hosts 文件，在文件最后添加一行。

```
192.168.100.10 controller
```

（3）安装基本服务

① 安装 MySQL（关系型数据库管理系统）服务。关于安装数据库的操作已经写成脚本，可以使用 iaas-install-mysql.sh 脚本来一键安装。

笔记

```
[root@ controller ~]# iaas-install-mysql.sh
```

② 安装 Keystone 服务。关于安装 Keystone 服务的操作，已经写成了脚本，可以通过使用 iaas-install-keystone.sh 脚本来进行一键安装。

```
[root@ controller ~]# iaas-install-keystone.sh
```

③ 安装 Glance 镜像服务。关于安装 Glance 服务的操作，已经写成了脚本，可以通过使用 iaas-install-glance.sh 脚本来进行一键安装。

```
[root@ controller ~]# iaas-install-glance.sh
```

④ 安装 Nova 计算服务。关于安装 Nova 服务的操作，已经写成了脚本，可以通过使用 iaas-install-nova-controller.sh 和 iaas-install-nova-compute.sh 这两个脚本来进行安装。

```
[root@ controller ~]# iaas-install-nova-controller.sh
[root@ controller ~]# iaas-install-nova-compute.sh
```

⑤ 安装 Neutron 网络服务。关于安装 Neutron 服务的操作，已经写成了脚本，可以通过使用 iaas-install-neutron-controller.sh 和 iaas-install-neutron-compute.sh 这两个脚本来进行安装。

```
[root@ controller ~]# iaas-install-neutron-controller.sh
[root@ controller ~]# iaas-install-neutron-compute.sh
```

这边的网络选择 gre 的方式，已经写成了脚本，可以通过 iaas-install-neutron-con-

troller-gre. sh 和 iaas-install-neutron-compute-gre. sh 这两个脚本来进行安装。

```
[root@ controller ~]# iaas-install-neutron-controller-gre. sh
[root@ controller ~]# iaas-install-neutron-compute-gre. sh
```

⑥ 安装 Dashboard 服务。关于安装 Dashboard 服务的操作，已经写成了脚本，可以通过使用 iaas-install-dashboard. sh 脚本来安装。

```
[root@ controller ~]# iaas-install-dashboard. sh
```

安装完以上的服务，实验环境搭建完成。

⑦ 验证安装。在浏览器中输入“http://192. 168. 100. 10/dashboard”，进入 OpenStack 登录界面，如图 3-1 所示。在“用户名”文本框中输入“admin”，在“密码”文本框中输入“000000”，输入完毕后单击“登入”按钮。

图 3-1 登录界面

项目实训

【实训题目】

使用 CentOS 6. 5 系统搭建一个 All-in-One 的 OpenStack 私有云平台。

【实训目的】

1. 掌握 Linux 系统的基础操作，包括修改主机名和配置网络等。
2. 掌握 OpenStack 私有云平台的搭建。

【实训内容】

1. 创建一个 CentOS 6. 5 的虚拟机修改基础的配置。

2. 执行安装脚本，搭建 OpenStack 平台。
3. 登录 OpenStack 平台，验证成功。

任务 3.2　配置 OpenStack Swift 对象存储

任务描述

1. 了解 Swift 对象存储。
2. 了解 Swift 的使用环境。
3. 掌握 Swift 对象存储的构建和使用。

知识学习

1. Swift 基本概念

Swift 构筑在比较便宜的标准硬件存储基础设施之上，无须采用 RAID（磁盘冗余阵列），通过在软件层面引入一致性散列技术提高数据冗余性、高可用性和可伸缩性。它支持多租户模式、容器和对象读写操作，适合解决互联网的应用场景下非结构化数据存储问题。在 OpenStack 中，Swift 主要用于存储虚拟机镜像，用于 Glance 的后端存储。在实际运用中，Swift 的典型运用是网盘系统，代表是 Dropbox，存储类型大多为图片、邮件、视频、存储备份等静态资源。

笔记

Swift 不能像传统文件系统那样进行挂载和访问，只能通过 REST API 接口来访问数据，并且这些 API 与亚马逊的 S3 服务 API 是兼容的。Swift 不同于传统文件系统和实时数据存储系统，它适用于存储和获取一些静态的永久性的数据并在需要的时候进行更新。

在了解 Swift 服务之前，首先要明确 Account、Container 和 Object 3 个基本概念。

① Account（账号）。出于访问安全性考虑，使用 Swift 系统，每个用户必须有一个账号（Account）。只有通过 Swift 验证的账号才能访问 Swift 系统中的数据。提供账号验证的节点被称为 Account Server。Swift 中由 Swauth 提供账号权限认证服务。

用户通过账号验证后将获得一个验证字符串（authentication token），后续每次数据访问操作都需要传递这个字符串。

② Container（容器）。Swift 中的 Container 可以类比 Windows 操作系统中的文件夹或者 UNIX 类操作系统中的目录，用于组织管理数据，所不同的是 Container 不能嵌套。数据都以 Object 的形式存放在 Container 中。

③ Object（对象）。它是 Swift 中的基本存储单元。一个对象包含两部分，数据和元数据（metadata）。其中元数据包括对象所属 Container 名称，对象本身名称以及用户添加的自定义数据属性（必须是 key-value 格式）。

对象名称在 URL 编码后大小要求小于 1024 个字节。用户上传的对象最大是 5 GB，最小是 0B。用户可以通过 Swift 内建的对象支持技术获取超过 5 GB 的大对象。对象的元数据不能超过 90 个 key-value，并且这些属性的总大小不能超过 4 KB。

Account、Container、Object 是 Swift 系统中的 3 个基本概念，三者的层次关系是一个 Account 可以创建拥有任意多个 Container，一个 Container 中可以包含任意多个 Object。可以简单理解为一个租户拥有一个 Account，Account 下存放 Container，Container 下存储 Object。

在 Swift 系统中，集群被划分成多个区（zone），区可以是一个磁盘、一个服务器、一台机柜甚至一个数据中心，每个区中有若干个节点（Node）。Swift 将 Object 存储在节点（Node）上，每个节点都是由多个硬盘组成的，并保证对象在多个节点上有备份（默认情况下，Swift 会给所有数据保存 3 个副本）以及这些备份之间的一致性。备份将均匀地分布在集群服务器上，并且系统保证各个备份分布在不同区的存储设备上，这样可以提高系统的稳定性和数据的安全性。它可以通过增加节点来线性地扩充存储空间。当一个节点出现故障，Swift 会从其他正常节点对出故障节点的数据进行备份。

2. Swift 服务架构

Swift 集群主要包含认证节点、代理节点和存储节点。认证节点主要负责对用户的请求授权，只有通过认证节点授权的用户才能操作 Swift 服务。Swift 是 OpenStack 的子项目之一，目前一般用 Keystone 服务作为 Swift 服务的认证服务。代理节点用于和用户交互，接受用户的请求并且给用户做出响应。Swift 服务所存储的数据一般都放在数据节点中。

如图 3-2 所示是 Swift 架构图，通过该图片本教材将详细讲解 Swift 服务的各个组件以及其功能。

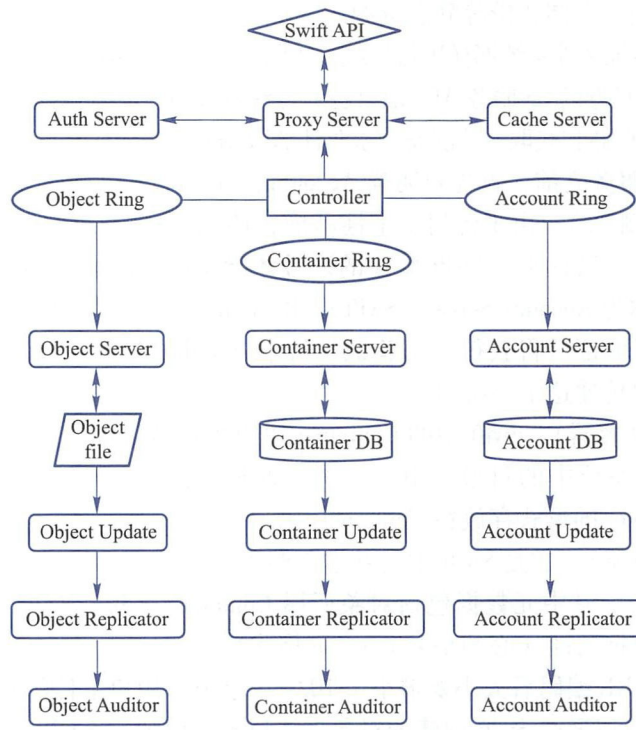


图 3-2 Swift 系统架构

笔记

① 代理服务（Proxy Server）。它对外提供对象服务 API，会根据环的信息来查找服务地址并转发用户请求至相应的账户、容器或者对象服务。由于采用无状态的 REST 请求协议，可以进行横向扩展来均衡负载。

② 认证服务（Authentication Server）。它用来验证访问用户的身份信息，并获得一个对象访问令牌（Token），在一定的时间内会一直有效。验证访问令牌的有效性缓存下来直至过期时间。

③ 缓存服务（Cache Server）。它缓存的内容包括对象服务令牌、账户和容器的存在信息，但不会缓存对象本身的数据。缓存服务可采用 Memcached（高性能的分布式内存对象缓存系统）集群，Swift 会使用一致性散列算法来分配缓存地址。

④ 账户服务（Account Server）。它提供账户元数据和统计信息，并维护所含容器列表的服务，每个账户的信息被存储在一个 SQLite（轻型数据库）数据库中。

⑤ 容器服务（Container Server）。它提供容器元数据和统计信息，并维护所含对象列表的服务，每个容器的信息也存储在一个 SQLite 数据库中。

⑥ 对象服务（Object Server）。它提供对象元数据和内容服务，每个对象的内容会以文件的形式存储在文件系统中，元数据会作为文件属性来存储，建议采用支持扩展属性的 XFS 文件系统。

⑦ 复制服务（Replicator）。它会检测本地分区副本和远程副本是否一致，具体是通过对比散列文件和高级水印来完成，发现不一致时会采用推式（Push）更新远程副本，如对象复制服务会使用远程文件拷贝工具 rsync 来同步。另外一个任务是确保被标记删除的对象从文件系统中移除。

⑧ 更新服务（Updater）。当对象由于高负载的原因而无法立即更新时，任务将会被序列化到在本地文件系统中进行排队，以便服务恢复后进行异步更新。如成功创建对象后容器服务器没有及时更新对象列表，这个时候容器的更新操作就会进入排队中，更新服务会在系统恢复正常后扫描队列并进行相应的更新处理。

⑨ 审计服务（Auditor）。它用来检查对象、容器和账户的完整性，如果发现比特级的错误，文件将被隔离，并复制其他的副本以覆盖本地损坏的副本。其他类型的错误会被记录到日志中。

⑩ 账户清理服务（Account Reaper）。它用来移除被标记为删除的账户，删除其所包含的所有容器和对象。

⑪ 环（Ring）。它是 Swift 最重要的组件，用于记录存储对象与物理位置间的映射关系。在涉及查询 Account、Container、Object 信息时，就需要查询集群的 Ring 信息。Ring 使用 Zone（区域）、Device（设备）、Partition（分区）和 Replica（副本）来维护这些映射信息。Ring 中每个 Partition 在集群中都（默认）有 3 个 Replica。每个 Partition 的位置由 Ring 来维护，并存储在映射中。Ring 文件在系统初始化时创建，之后每次增减存储节点时，需要重新平衡一下 Ring 文件中的项目，以保证增减节点时，系统因此而发生迁移的文件数量最少。

⑫ 区域（Zone）。Ring 中引入了 Zone 的概念，把集群的 Node 分配到每个 Zone 中。其中同一个 Partition 的 Replica 不能同时放在同一个 Node 上或同一个 Zone 内。防

止造成所有的 Node 如果都在一个机架或一个机房中，一旦发生断电、网络故障等，造成用户无法访问的情况出现。

任务实施

① 安装 Swift 服务。在 controller 节点依次执行 iaas-install-swift-controller.sh 和 iaas-install-swift-compute.sh 脚本即可完成安装。

```
[root@controller ~]# iaas-install-swift-controller.sh
[root@controller ~]# iaas-install-swift-compute.sh
```

安装完 Swift 之后，查看一下 Swift 的状态。

```
[root@controller ~]# swift stat
          Account: AUTH_d7ccaa2d63a54513a15faf9927d8c39f
       Containers: 0
          Objects: 0
            Bytes: 0
X-Put-Timestamp: 1494617702.20632
     X-Timestamp: 1494617702.20632
      X-Trans-Id: txb4416f0324634ead98ac4-0059160e65
    Content-Type: text/plain; charset=utf-8
```

查看容器。

```
[root@controller ~]# swift list
```

仓库为空，因为没有容器，所以查询不到。

② 创建容器。创建一个容器，名称为 gw001，并查看。

```
[root@controller ~]# swift post gw001
[root@controller ~]# swift list
gw001
```

③ 容器操作。

上传一个文件到这个容器中，并查看。

```
[root@controller ~]# swift upload gw001 anaconda-ks.cfg
anaconda-ks.cfg
[root@controller ~]# swift list gw001
anaconda-ks.cfg
```

删除该文件并查看。

```
[root@controller ~]# swift delete gw001 anaconda-ks.cfg
anaconda-ks.cfg
```

```
[root@ controller ~]# swift list gw001
```

可以发现文件已被删除。

删除该容器，并查看。

```
[root@ controller ~]# swift delete gw001
gw001
[root@ controller ~]# swift list
```

可以发现容器已被删除。

项目实训

【实训题目】

部署 Swift 服务，对 Swift 存储进行上传文件和删除操作。

【实训目的】

1. 掌握 Swift 对象存储的搭建。
2. 掌握 Swift 对象存储的使用和运维。

【实训内容】

1. 使用安装脚本安装 Swift 对象存储服务。
2. 查看 Swift 状态，并创建一个名称为 test 的容器。
3. 创建一个 1. txt 文档并上传到 test 容器中。

任务 3.3　配置 OpenStack 分布式块存储

任务描述

1. 了解 Cinder 块存储。
2. 了解 Cinder 块存储的使用环境。
3. 掌握 Cinder 块存储的搭建和使用。

知识学习

1. 了解 Cinder 块存储

（1）块级

块是指以扇区为基础，一个或多个连续的扇区组成的存储结构，也叫物理块。它是处在文件系统与块设备（如磁盘驱动器）之间的一种概念。

（2）块存储

Cinder 提供了 OpenStack 的 Block Service（块服务）。类似于 Amazon 的 EBS 块存储服务，OpenStack 中的实例是不能持久化的，需要挂载 Volume，在 Volume 中实现持

笔记

久化。Cinder 就是对 Volume 的管理。它是从 Nova 里分出来的，前身是 Nova-Volume。因为 Nova 变得越来越复杂，而块服务又那么重要，所以在 Folsom 版本中，Cinder 就从 Nova 中分离出来了。

2. Cinder 架构图

Cinder 的架构如图 3-3 所示。

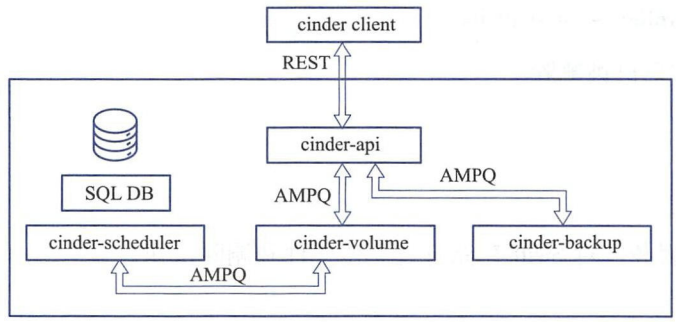


图 3-3 Cinder 构架

（1）cinder-api

它负责接受和处理 REST 请求，并将请求放入 RabbitMQ 队列。

（2）cinder-scheduler

它用来处理任务队列的任务，并根据预定策略选择合适的 Volume Service 节点来执行任务。目前版本的 Cinder 仅仅提供了一个 Simple Scheduler，该调度器选择卷数量最少的一个活跃节点来创建卷。

（3）cinder-volume

该服务运行在存储节点上，管理存储空间。每个存储节点都有一个 Volume Service，若干个这样的存储节点联合起来就可以构成一个存储资源池。为了支持不同类型和型号的存储，当前版本的 Cinder 为 Volume Service 添加下列 drivers。

① 本地存储：LVM（iSCSI）、Sheepdog（sheepdog）。

② 网络存储：NFS、RBD（Ceph）。

③ IBM：Storwize family/SVC（iSCSI/FC）、XIV（iSCSI）、GPFS、ZVM。

④ Netapp：NetApp（iSCSI/NFS）。

⑤ EMC：VMAX/VNX（iSCSI）、Isilon（iSCSI）。

⑥ Solidfire：Solidfire cluster（iSCSI）。

⑦ HP：3PAR（iSCSI/FC）、LeftHand（iSCSI）。

当然在 Cinder 的 blueprints（蓝图）当中还有一些其他的 drivers，以后的版本可能会添加进来。

任务实施

（1）环境搭建

利用上节已经搭建完毕的 IaaS 平台的一个分区（本任务使用的是/dev/sda4）。

在 controller 节点执行下列脚本，按顺序安装 Cinder 服务。

```
# iaas-install-cinder-controller.sh
# iaas-install-cinder-compute.sh
```

（2）使用 Cinder 块存储

① 登录 OpenStack。执行完脚本，在浏览器中输入“http://192.168.100.10/dashboard”，进入 OpenStack 的登录界面。在“用户名”文本框中输入“admin”，在“密码”文本框中输入“000000”，输入完毕后单击“登入”按钮。

② 修改安全规则。在界面左侧选择“项目”→“Compute”→“访问 & 安全”选项卡，接着在界面右侧单击“管理规则”按钮进行安全规则的修改，如图 3-4 所示。

图 3-4　访问 & 安全

进入安全组规则页面，单击右上角“添加规则”按钮，如图 3-5 所示，进入添加规则页面。

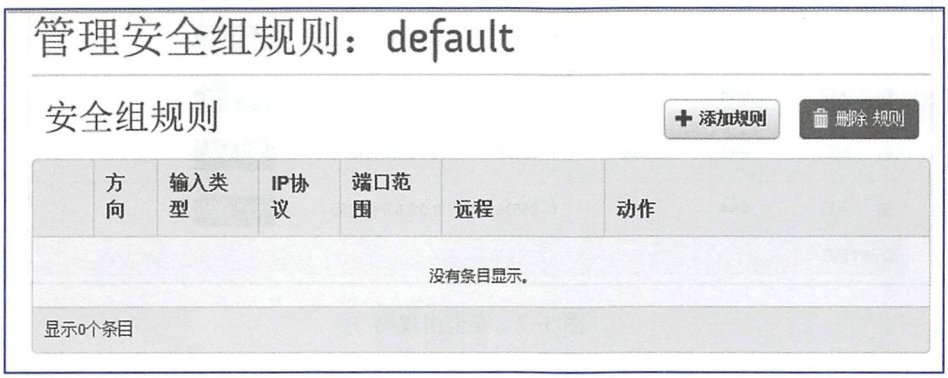


图 3-5　修改安全规则

单击“规则”下的下拉▼按钮，在弹出的下拉菜单中选择“ALL ICMP”选项；单击“方向”下的下拉▼按钮，在弹出的下拉菜单中选择“入口”选项；单击“远程”下的下拉▼按钮，在弹出的下拉菜单中选择“CIDR”选项；在“CIDR”文本框中输入“0.0.0.0/0”，如图 3-6 所示。填写完成后单击右下角的“添加”按钮。

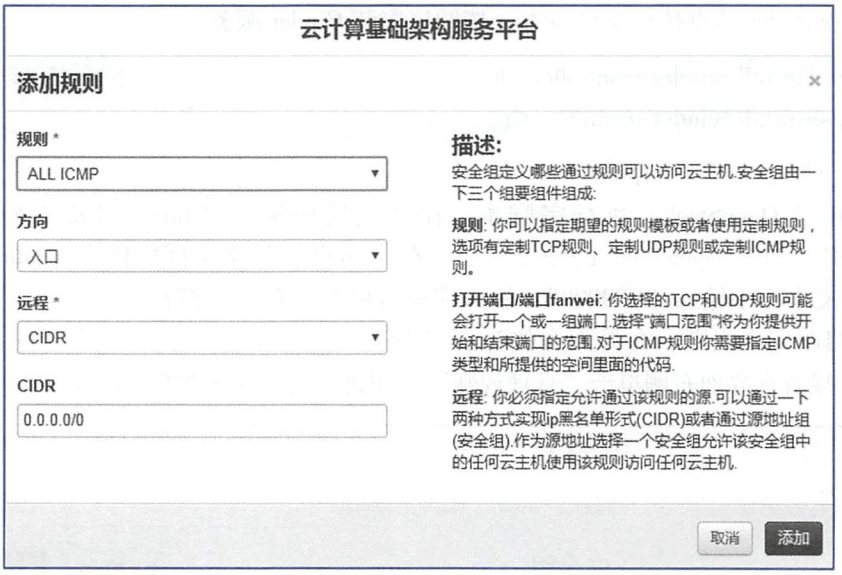


图 3-6　添加规则

添加所有 IMCP、TCP、UDP 的入口和出口规则，添加完之后如图 3-7 所示。

图 3-7　安全组规则

笔 记

③ 创建网络。在界面左侧选择“管理员”→“系统面板”→“网络”选项卡，接着在界面右侧单击“创建网络”按钮，如图 3-8 所示。

首先创建一个网络（外网），在打开的“云计算基础架构服务平台”对话框的“名称”文本框中输入“ext-net”，单击“项目”下的下拉按钮▼，在弹出的下拉菜单中选择“admin”选项，选中“管理员状态”“共享的”和“外部网络”3 个复选项，最后单击右下角的“创建网络”按钮，完成创建，如图 3-9 所示。

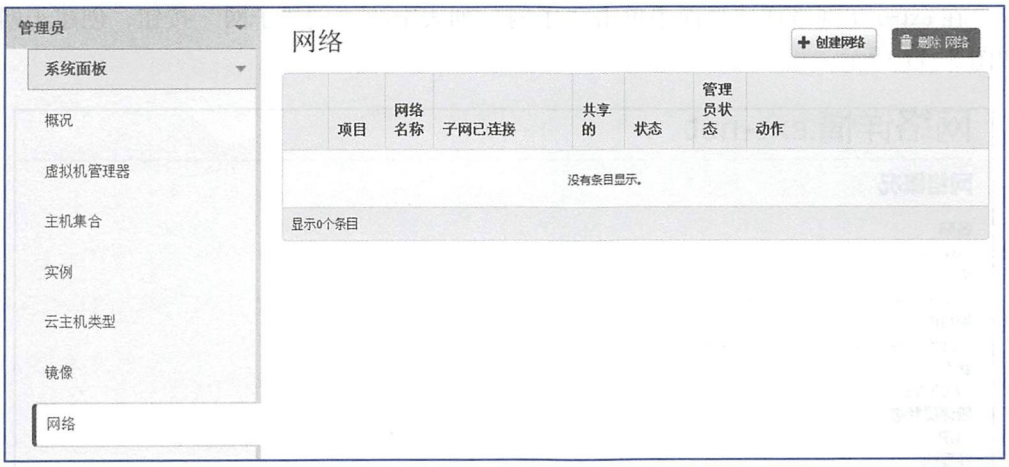


图 3-8　网络列表

图 3-9　创建外网

在“网络”列表中选择网络名称 ext-net，如图 3-10 所示，进入该网络。

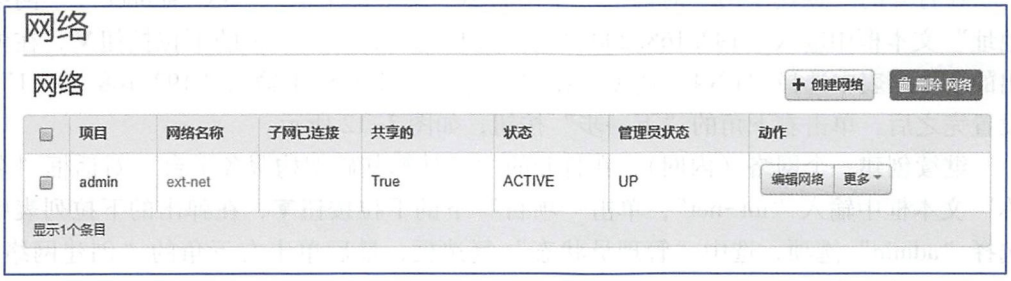


图 3-10　进入网络

在 ext-net 网络详情页面中单击“子网”列表中的“创建子网”按钮，创建子网，如图 3-11 所示。

图 3-11　创建子网

在打开的“创建子网”对话框“子网名称”文本框中输入“ext-subnet”，“网络地址”文本框中输入“192. 168. 200. 0/24”，单击“IP 版本”下的下拉按钮▼，在弹出的下拉列表中选择“IPv4”选项，在“网管 IP”文本框中输入“192. 168. 200. 1”。设置完之后，单击右下角的“下一步”按钮，如图 3-12 所示。

继续创建一个网络（内网），在打开的“云计算基础架构服务平台”对话框“名称”文本框中输入“int-net”，单击“项目”下的下拉按钮▼，在弹出的下拉列表中选择“admin”选项，选中“管理员状态”复选框，最后单击右下角的“创建网络”按钮，完成创建，如图 3-13 所示。

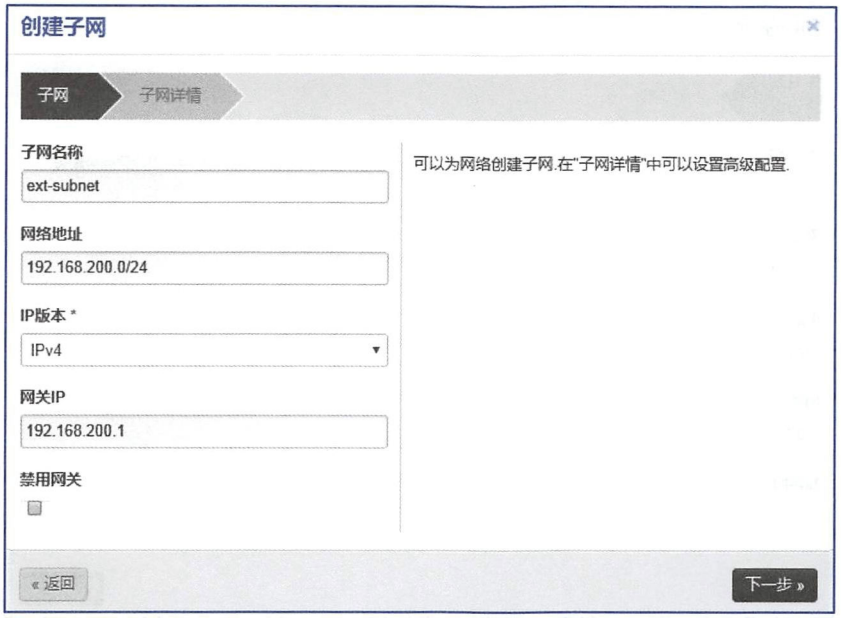


图 3-12　创建子网信息

图 3-13　创建内网信息

为刚才创建的内网添加一个子网。进入 int-net 网络详情页面，单击“子网”列表右侧的“创建子网”按钮。在打开的“创建子网”对话框的“子网名称”文本框中输入“int-subnet”，在“网络地址”文本框中输入“10.0.0.0/24”，单击“IP 版本”下的下拉按钮▼，在弹出的下拉列表中选择“IPv4”选项，在“网管 IP”文本框中输入“10.0.0.1”。设置完之后，单击右下角的“下一步”按钮，如图 3-14 所示。

图 3-14　创建内网的子网

④ 创建路由。在界面左侧选择“项目”→“网络”→“路由”选项卡，接着在界面右侧“路由”列表中单击“新建路由”按钮，进行路由的创建，如图 3-15 所示。

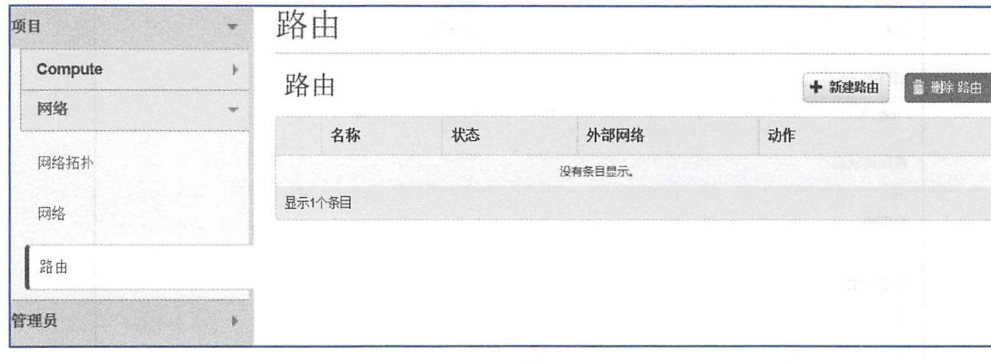


图 3-15　创建路由

在打开的“云计算基础架构服务平台”对话框“路由名称”文本框中输入“route”，然后单击右下角的“新建路由”按钮，如图 3-16 所示。

在“路由”列表中单击“设置网关”按钮，按提示配置网关，如图 3-17 所示。

选择路由名称 route，进入路由详情页面，在“接口”列表中单击“增加接口”按钮，为路由增加接口。在打开的“云计算基础架构服务平台”对话框中单击“子网”下的下拉按钮▼，在弹出的下拉列表中选择“int - net：10. 0. 0. 0/24（int - subnet）”选项，完成后单击右下角“增加接口”按钮，如图 3-18 所示。

笔记

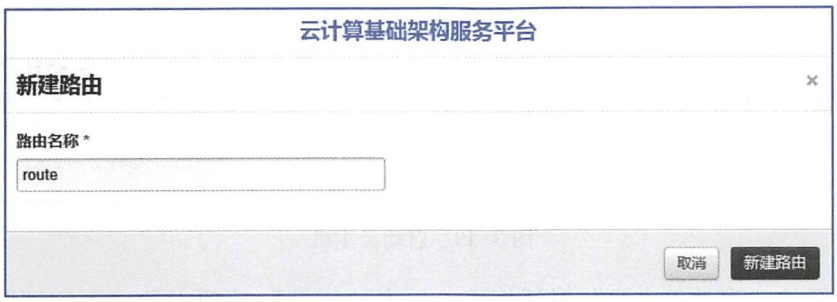


图 3-16　创建路由信息

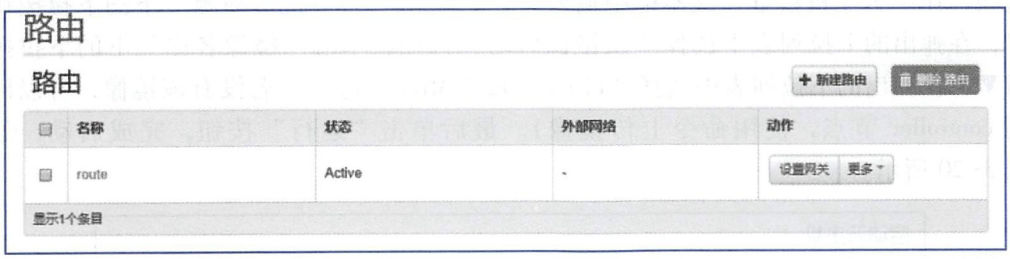


图 3-17　设置网关

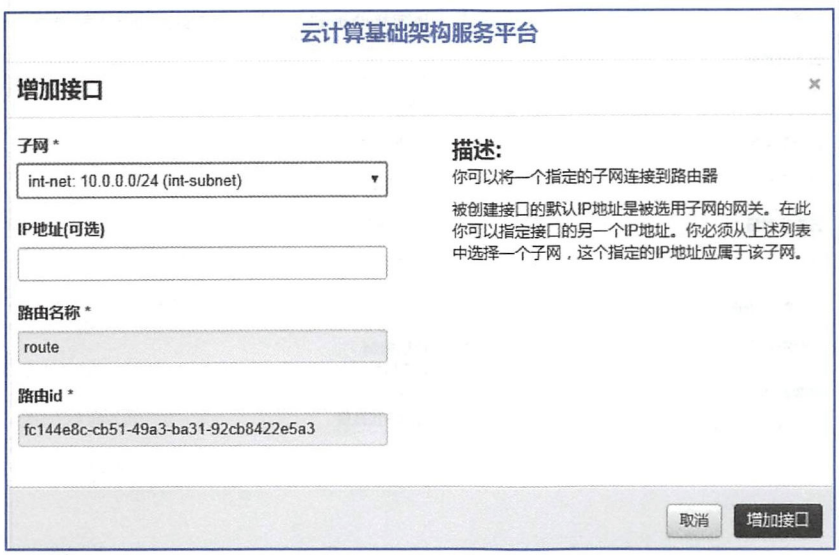


图 3-18　配置增加接口的信息

⑤ 创建云主机。在界面左侧选择“项目”→“Compute”→“实例”选项卡，接着在界面右侧单击“启动云主机”按钮，如图 3-19 所示。

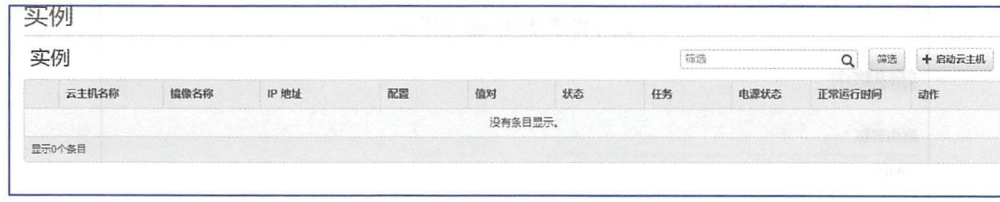


图 3-19　启动云主机

笔 记

在打开的“启动云主机”对话框中，单击“详情”选项卡的“可用域”下的下拉按钮▼，在弹出的下拉列表中选择“nova”选项，在“云主机名称”文本框中输入“test”，单击“云主机类型”下的下拉按钮▼，在弹出的下拉列表中选择“m1. tiny”选项，在“云主机数量”文本框中输入“1”，单击“云主机启动源”下的下拉按钮▼，在弹出的下拉列表中选择“从镜像启动”选项，单击“镜像名称”下的下拉按钮▼，在弹出的下拉列表中选择“cirros（12. 7 MB）”选项（若没有该镜像，可以回到 controller 节点，使用命令上传镜像）。最后单击“运行”按钮，完成启动，如图 3-20 所示。

图 3-20　云主机信息

选择“网络”选项卡，然后单击“int-net”网络选项的+按钮，最后单击“运行”按钮，如图 3-21 所示。

图 3-21　选择网络

单击“运行”按钮后，在“实例”列表中会创建刚才配置的云主机，单击右侧的“绑定浮动 IP”按钮，如图 3-22 所示。

图 3-22　绑定浮动 IP

在打开的“管理浮动 IP 的关联”对话框中，单击“IP 地址”下+按钮，如图 3-23 所示。

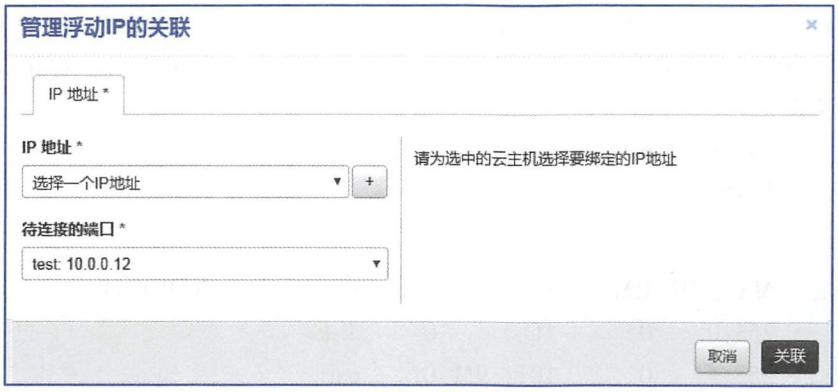


图 3-23　管理浮动 IP 的关联

在打开的“云计算基础构架服务平台”对话框中，保持默认，单击右下角“分配IP”按钮，如图3-24所示。

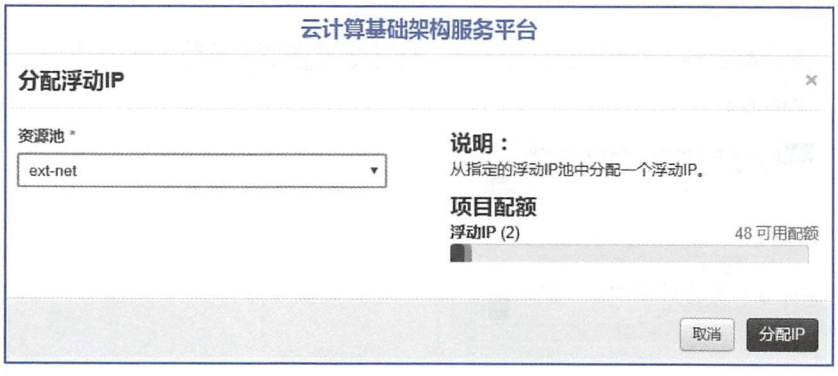


图3-24　分配IP

返回“管理浮动IP的关联”对话框，保持默认，单击“关联”按钮，关联浮动IP，如图3-25所示。

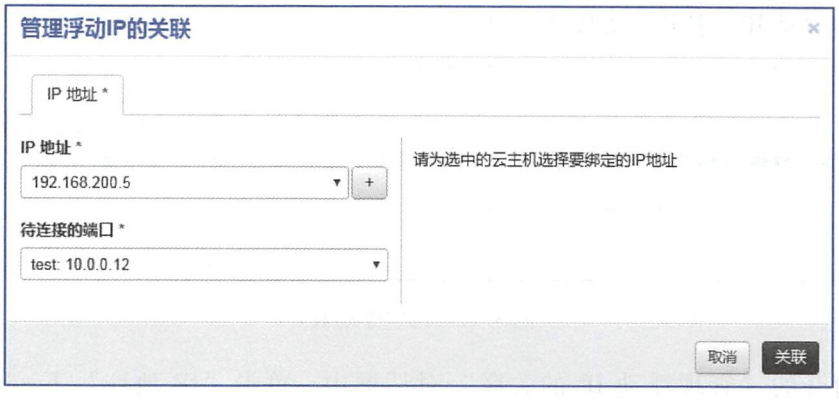


图3-25　关联浮动IP

笔记

⑥登录云主机。开发者可以在SecureCRT中输入创建的云主机的IP“192.168.200.5”、账号“cirros”、密码“cubswin:)”来登录云主机，登录完成后如图3-26所示。

到此为止即验证登录OpenStack成功。

查看云主机硬盘。

```
$lsblk
NAME    MAJ:MIN RM    SIZE    RO  TYPE  MOUNTPOINT
vda     253:0   0     1G      0   disk
'-vda1  253:1   0     1011.9M 0   part  /
```

可以发现主机有一个vda的硬盘。

⑦云硬盘挂载。返回到controller节点，创建一个卷设备，名称为test 1。

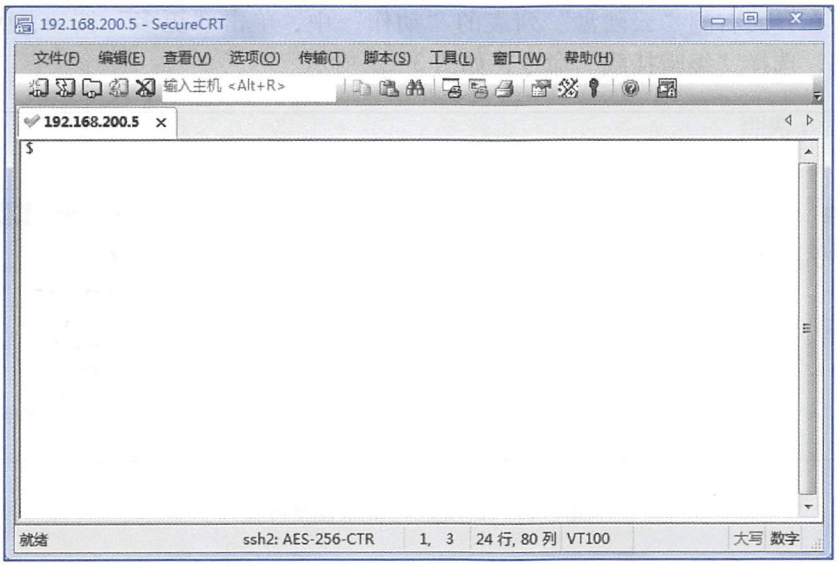


图 3-26　CRT 成功连接云主机

```
[root@controller ~]# cinder create --display-name test1 2
+---------------------+--------------------------------------+
|       Property      |                Value                 |
+---------------------+--------------------------------------+
|     attachments     |                  []                  |
|  availability_zone  |                 nova                 |
|      bootable       |                false                 |
|     created_at      |      2017-08-03T05:28:04.160666      |
| display_description |                 None                 |
|     display_name    |                test1                 |
|      encrypted      |                False                 |
|          id         | cb09e918-ebed-4b13-b4fc-41377ad05c3b |
|       metadata      |                  {}                  |
|         size        |                  2                   |
|     snapshot_id     |                 None                 |
|     source_volid    |                 None                 |
|        status       |               creating               |
|     volume_type     |                 None                 |
+---------------------+--------------------------------------+
```

上述代码的作用是创建一个名称为 test 1、大小为 2 GB 的卷。可以使用“cinder list”命令查看。

在 OpenStack 主页面中，在界面左侧选择“项目”→“Compute”→“云硬盘”

选项卡。在界面右侧“云硬盘”列表的“动作”中，单击“更多”按钮，弹出的下拉菜单中，选择“编辑挂载”命令，如图 3-27 所示。

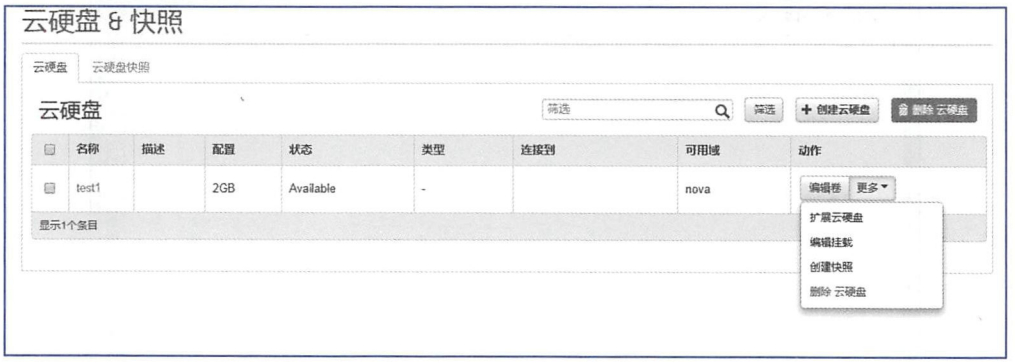


图 3-27　云硬盘 & 快照管理列表

笔 记

在打开的“云计算基础架构服务平台”对话框中，单击“连接到云主机”下的下拉按钮▼，在弹出的下拉列表中选择刚才创建的 test 主机，然后单击右下角的“连接云硬盘”按钮，如图 3-28 所示。

图 3-28　连接到云主机

然后在“云硬盘 & 快照”界面中可以看到“连接到”中有“在设备/dev/vdb 上连接到 test”的信息，如图 3-29 所示。

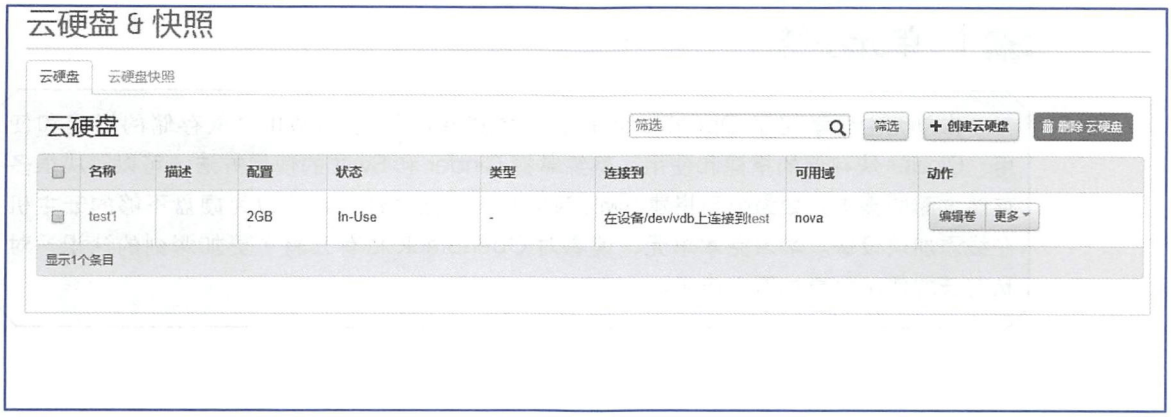


图 3-29　连接设备信息显示

到云主机上查看设备。

```
$lsblk
NAME    MAJ:MIN BRM    SIZE    RO    TYPE    MOUNTPOINT
vda     253:0    0     1G      0     disk
'-vda1  253:1    0     1011.9M 0     part    /
vdb     253:16   0     2G      0     disk
$
```

可以看到一个大小为 2 GB 的 vdb 硬盘。验证 Cinder 块存储服务成功。

项目实训

【实训题目】

创建一个云主机，并通过 Cinder 给云主机挂载 2 GB 的云硬盘。

【实训目的】

1. 掌握 OpenStack 的配置，包括访问安全规则、网络和路由。
2. 掌握如何使用 OpenStack 创建云主机。
3. 掌握 Cinder 块存储的创建和如何挂载到云主机上使用。

【实训内容】

1. 登录 OpenStack 的 Dashboard 并完成配置。
2. 创建一台云主机。
3. 创建一个 Cinder 块存储，大小为 2 GB。
4. 连接 Cinder 块存储到云主机，并使用该块设备。

单元小结

本单元主要学习了 OpenStack 私有云的搭建和使用、Swift 对象存储的搭建和使用、Cinder 块存储的搭建和使用。熟练掌握 Cinder 和 Swift 的使用方法，可以解决很多存储方面的需求，如为公司搭建 Swift 服务器来存储文件，也可以为硬盘不够的云主机在线添加块设备。学习完本单元，读者对 OpenStack 私有云有了更加深刻的认识，对私有云的使用会更加得心应手。

单元 4

构建GlusterFS的分布式文件系统

学习目标

【知识目标】

- 了解 GlusterFS 文件系统。
- 了解 GlusterFS 的使用环境和特点。
- 了解 GlusterFS 的搭建和使用。

【技能目标】

- 掌握 GlusterFS 文件系统的搭建和使用。
- 掌握 GlusterFS 文件系统的简单运维。

PPT4：构建 GlusterFS 的分布式文件系统

学习情境

研发部采用小缪实行数据集中存储的方案进行建设。因为可供选择的后端集群存储方式比较多，研发部对这方面并没有什么经验，于是安排小缪选择 GlusterFS 文件系统，按照实际环境要求进行部署，并测试使用。

（1）项目设计

使用两台服务器搭建一个 GlusterFS 文件系统，并挂载使用。

（2）服务器功能

① CentOS 6.5 实现 GlusterFS 集群的搭建和使用。

② 实现基于 GlusterFS 文件系统的集群存储。

任务 4　搭建 GlusterFS 文件系统

任务描述

1. 了解 GlusterFS 文件系统的架构。
2. 了解 GlusterFS 文件系统的特点。
3. 掌握 GlusterFS 文件系统的搭建和使用。

知识学习

笔 记

1. GlusterFS 文件系统概述

GlusterFS 是 Scale-Out 存储解决方案 Gluster 的核心，它是一个开源的分布式文件系统，具有强大的横向扩展能力，通过扩展能够支持数 PB（petabyte）存储容量和处理数千客户端。GlusterFS 借助 TCP/IP 或 InfiniBand RDMA（一种支持多并发链接的“转换线缆”技术）网络将物理分布的存储资源聚集在一起，使用单一全局命名空间来管理数据。GlusterFS 基于可堆叠的用户空间设计，可为各种不同的数据负载提供优异的性能。

GlusterFS 支持任何运行在标准 IP 网络上的标准应用程序和标准客户端，如图 4-1 所示，用户可以在全局统一的命名空间中使用 NFS/CIFS 等标准协议来访问应用数据。GlusterFS 使得用户可摆脱原有的独立、高成本的封闭存储系统，能够利用普通廉价的存储设备来部署可集中管理、横向扩展、虚拟化的存储池，其存储容量可扩展至 TB/PB 级。GlusterFS 主要特征如下：

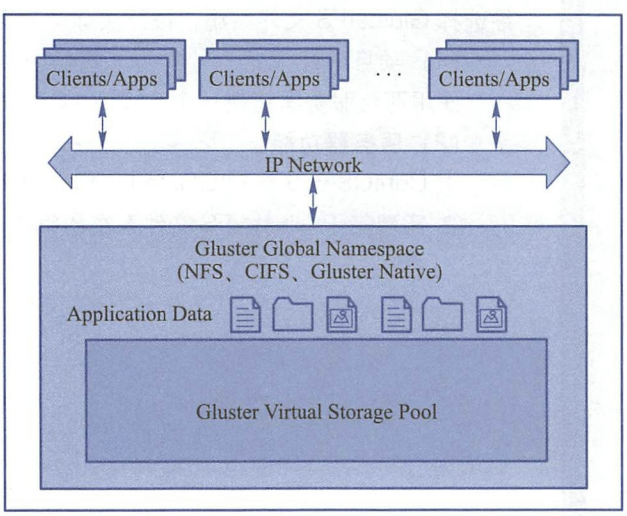


图 4-1　GlusterFS 网络协议

（1）扩展性和高性能

GlusterFS 利用双重特性来提供几 TB 至数 PB 的高扩展存储解决方案。Scale-Out（横向扩展）架构允许通过简单地增加资源的方法来提高存储容量和性能，磁盘、计算和 I/O 资源都可以独立增加，支持 10 GbE（千兆以太网）和 InfiniBand（无限带宽技术）等高速网络互联。Gluster 弹性哈希（Elastic Hash）解除了 GlusterFS 对元数据服务器的需求，消除了单点故障和性能瓶颈，真正实现了并行化数据访问。

（2）高可用性

GlusterFS 可以对文件进行自动复制，如镜像或多次复制，从而确保数据总是可以访问的，甚至是在硬件故障的情况下也能正常访问。自我修复功能能够把数据恢复到正确的状态，而且修复是以增量的方式在后台执行，几乎不会产生性能负载。GlusterFS 没有设计自己的私有数据文件格式，而是采用操作系统中主流标准的磁盘文件系统（如 Ext 3、ZFS）来存储文件，因此数据可以使用各种标准工具进行复制和访问。

（3）全局统一命名空间

全局统一命名空间将磁盘和内存资源聚集成一个单一的虚拟存储池，对上层用户和应用屏蔽了底层的物理硬件。存储资源可以根据需要在虚拟存储池中进行弹性扩展，如扩容或收缩。当存储虚拟机镜像时，存储的虚拟镜像文件没有数量限制，成千上万的虚拟机均通过单一挂载点进行数据共享。虚拟机 I/O 可在命名空间内的所有服务器上自动进行负载均衡，消除了 SAN 环境中经常发生的访问热点和性能瓶颈问题。

笔记

（4）弹性哈希算法

GlusterFS 采用弹性哈希算法在存储池中定位数据，而不是采用集中式或分布式元数据服务器索引。在其他的 Scale-Out 存储系统中，元数据服务器通常会导致 I/O 性能瓶颈和单点故障问题。在 GlusterFS 中，所有在 Scale-Out 存储配置中的存储系统都可以智能地定位任意数据分片，不需要查看索引或者向其他服务器查询。这种设计机制完全并行化了数据访问，实现了真正的线性性能扩展。弹性卷管理数据储存在逻辑卷中，逻辑卷可以从虚拟化的物理存储池进行独立逻辑划分而得到。存储服务器可以在线进行增加和移除，不会导致应用中断。逻辑卷可以在所有配置服务器中增加和缩减，可以在不同服务器迁移进行容量均衡，或者增加和移除系统，这些操作都可在线进行。文件系统配置更改也可以实时在线进行并应用，从而可以适应工作负载条件变化或在线性能调优。

（5）基于标准协议

Gluster 存储服务支持 NFS、CIFS、HTTP、FTP 以及 Gluster 原生协议，完全与 POSIX（Portable Operating System Interface of UNIX，可移植操作系统接口）标准兼容。现有应用程序不需要作任何修改或使用专用 API，就可以对 Gluster 中的数据进行访问。这在公有云环境中部署 Gluster 时非常有用，Gluster 对云服务提供商专用 API 进行抽象，然后提供标准 POSIX 接口。

2. GlusterFS 文件系统的特点

（1）设计目标

GlusterFS 的设计思想显著区别现有并行、集群、分布式文件系统。如果 GlusterFS 在设计上没有本质性的突破，则其难以在与 Lustre（平行分布式文件系统）、PVFS2、Ceph（分布式文件系统）等的竞争中占据优势，更无法与 GPFS（General Parallel File System，共享文件系统）、StorNext、ISILON、IBRIX 等具有多年技术沉淀和市场积累的商用文件系统竞争。其核心设计目标如下：

① 弹性存储系统（Elasticity）。存储系统具有弹性能力，意味着企业可以根据业务需要灵活地增加或缩减数据存储以及增删存储池中的资源，而不需要中断系统运行。GlusterFS 设计目标之一就是弹性，允许动态增删数据卷、扩展或缩减数据卷、增删存储服务器等，不影响系统正常运行和业务服务。GlusterFS 早期版本中弹性不足，部分管理工作需要中断服务，目前最新版本的 GlusterFS 3.1.X 已经弹性十足，能够满足对存储系统弹性要求高的应用需求，尤其是对云存储服务系统而言意义更大。GlusterFS 主要通过存储虚拟化技术和逻辑卷管理来实现这一设计目标。

② 线性横向扩展（Linear Scale-Out）。线性扩展对于存储系统而言是非常难实现的，通常系统规模扩展与性能提升之间是 LOG 对数曲线关系，因为同时产生的相应负载而消耗了部分性能的提升。现在很多并行、集群、分布式文件系统都具有很高的扩展能力，Luster 存储节点可以达到 1,000 个以上，客户端数量能够达到 25,000 个以上，这个扩展能力是非常强大的，但是 Lustre 也不是线性扩展的。

③ 纵向扩展（Scale-Up）。旨在提高单个节点的存储容量或性能，往往存在理论上或物理上的各种限制而无法满足存储需求。Scale-Out 通过增加存储节点来提升整个系统的容量或性能，这一扩展机制是目前存储技术热点，能有效应对容量、性能等存储需求。目前的并行、集群、分布式文件系统大多都具备横向扩展能力。GlusterFS 是线性横向扩展架构，它通过扩展存储节点的数量即可获得线性增长的存储容量和性能的提升。因此，结合纵向扩展 GlusterFS 可以获得多维扩展能力，增加每个节点的磁盘可增加存储容量，增加存储节点可以提高性能，从而将更多磁盘、内存、I/O 资源聚集成更大容量、更高性能的虚拟存储池。GlusterFS 利用以下 3 种基本技术来获得线性横向扩展能力。

- 消除元数据服务。
- 高效数据分布，获得扩展性和可靠性。
- 通过完全分布式架构的并行化获得性能的最大化。

（2）高可靠性（Reliability）

与 GFS（Google File System）类似，GlusterFS 可以构建在普通的服务器和存储设备之上，因此可靠性显得尤为关键。GlusterFS 从设计之初就将可靠性纳入核心设计，采用了多种技术来实现这一设计目标。首先，它假设故障是正常事件，包括硬件、磁盘、网络故障以及管理员误操作造成的数据损坏等。GlusterFS 设计支持自动复制和自动修复功能来保证数据的可靠性，不需要管理员的干预。其次，GlusterFS 利用了底层

笔记

Ext 3 或 ZFS 等磁盘文件系统的日志功能来提供一定的数据可靠性，而没有自己重新开发底层文件系统。再次，GlusterFS 是无元数据服务器设计的，不需要元数据的同步或者一致性维护，很大程度上降低了系统的复杂性，不仅提高了性能，还大大地提高了系统的可靠性。

（3）技术特点

GlusterFS 在技术实现上与传统存储系统或现有其他分布式文件系统有显著不同之处，主要体现在以下几个方面。

① 完全软件实现（Software Only）。GlusterFS 认为存储是软件问题，不能把用户局限于使用特定的供应商或硬件配置来解决。GlusterFS 采用开放式设计，广泛支持工业标准的存储、网络和计算机设备，而非与定制化的专用硬件设备捆绑。对于商业客户，GlusterFS 可以以虚拟装置的形式交付，也可以与虚拟机容器打包，或者是公有云中部署的镜像。开源社区中，GlusterFS 被大量部署在基于廉价闲置硬件的各种操作系统上，构成集中统一的虚拟存储资源池。简而言之，GlusterFS 是通过开放的全软件来实现，完全独立于硬件和操作系统。

② 完整的存储操作系统栈（Complete Storage Operating System Stack）。GlusterFS 不仅提供了一个分布式文件系统，而且还提供了许多其他重要的分布式功能，如分布式内存管理、I/O 调度、软 RAID 和自我修复等。GlusterFS 汲取了微内核架构的经验教训，借鉴了 GNU/Hurd 操作系统的设计思想，在用户空间实现了完整的存储操作系统栈。

③ 用户空间实现（User Space）。与传统的文件系统不同，GlusterFS 在用户空间实现，这使得其安装和升级特别简便。另外，这也极大降低了普通用户修改 GlusterFS 源码的门槛，仅仅需要通用的 C 程序设计技能，而不需要特别的内核编程经验。

④ 模块化堆栈式架构（Modular Stackable Architecture）。GlusterFS 采用模块化、堆栈式的架构，可通过灵活的配置支持高度定制化的应用环境，如大文件存储、海量小文件存储、云存储、多传输协议应用等。每个功能以模块形式实现，然后以积木方式进行简单的组合，即可实现复杂的功能。例如，Replicate 模块可实现 RAID 1，Stripe 模块可实现 RAID 0，通过两者的组合可实现 RAID 10 和 RAID 01，同时获得高性能和高可靠性。

⑤ 原始数据格式存储（Data Stored in Native Formats）。GlusterFS 以原始数据格式（如 Ext 3、Ext 4、XFS、ZFS）储存数据，并实现多种数据自动修复机制。因此，系统极具弹性，即使离线情形下文件也可以通过其他标准工具进行访问。如果用户需要从 GlusterFS 中迁移数据，不需要作任何修改仍然可以使用这些数据。

⑥ 无元数据服务设计（No Metadata with the Elastic Hash Algorithm）。对 Scale-Out 存储系统而言，最大的挑战之一就是记录数据逻辑与物理位置的映像关系，即数据元数据，可能还包括诸如属性和访问权限等信息。传统分布式存储系统使用集中式或分布式元数据服务来维护元数据，集中式元数据服务会导致单点故障和性能瓶颈问题，而分布式元数据服务存在性能负载和元数据同步一致性问题。特别是对于海量小文件的应用，元数据问题是个非常大的挑战。

GlusterFS 独特地采用无元数据服务的设计，取而代之使用算法来定位文件，元数据和数据没有分离而是一起存储。集群中的所有存储系统服务器都可以智能地对文件数据分片进行定位，仅仅根据文件名和路径并运用算法即可，而不需要查询索引或者其他服务器。这使得数据访问完全并行化，从而实现真正的线性性能扩展。无元数据服务器极大地提高了 GlusterFS 性能、可靠性和稳定性。其总体架构与设计如图 4-2 所示。

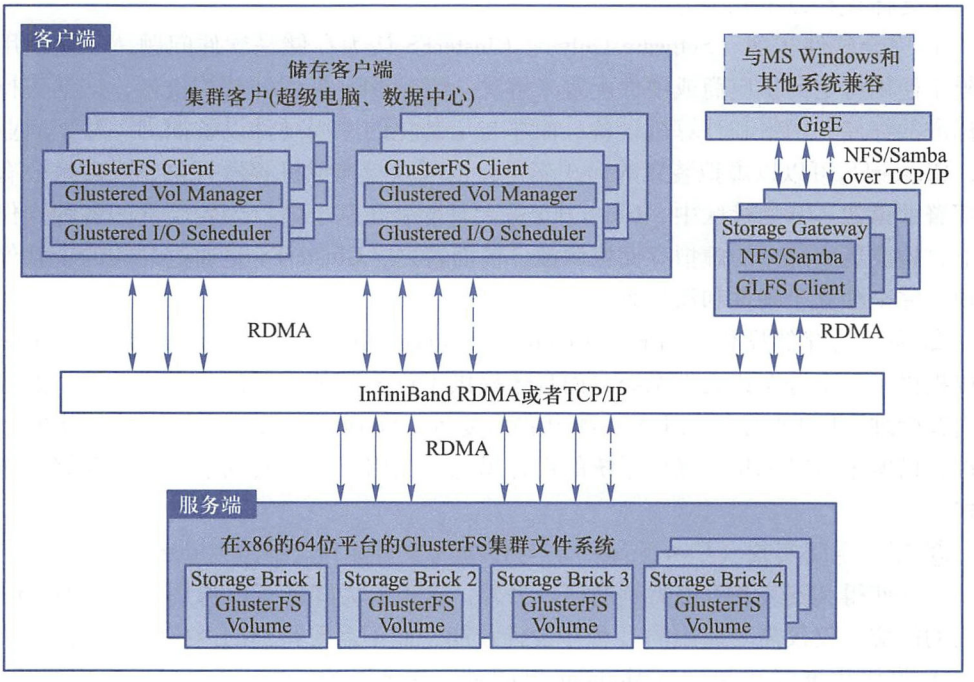


图 4-2　GlusterFS 架构与设计

笔记

GlusterFS 总体架构与组成部分主要由存储服务器（Brick Server）、客户端以及 NFS/Samba 存储网关组成。不难发现，GlusterFS 架构中没有元数据服务器组件，这是其最大的设计特点，对于提升整个系统的性能、可靠性和稳定性都有着决定性的意义。GlusterFS 支持 TCP/IP 和 InfiniBand RDMA 高速网络互联，客户端可通过原生的 GlusterFS 协议访问数据，其他没有运行 GlusterFS 客户端的终端可使用 NFS/CIFS 标准协议通过存储网关访问数据。

存储服务器主要提供基本的数据存储功能，最终的文件数据通过统一的调度策略分布在不同的存储服务器上。它们上面运行着 Glusterfsd 来负责处理来自其他组件的数据服务请求。如前所述，数据以原始格式直接存储在服务器的本地文件系统上，如 Ext 3、Ext 4、XFS、ZFS 等，运行服务时指定数据存储路径。多个存储服务器可以通过客户端或存储网关上的卷管理器组成集群，如 Stripe（RAID 0）、Replicate（RAID 1）和 DHT（分布式 Hash）存储集群，也可利用嵌套组合构成更加复杂的集群，如 RAID 10。

由于没有了元数据服务器，客户端承担了更多的功能，包括数据卷管理、I/O 调

度、文件定位、数据缓存等功能。客户端上运行 Glusterfs 进程，它实际是 Glusterfsd 的符号链接，利用 FUSE（File system in User Space）模块将 GlusterFS 挂载到本地文件系统之上，实现 POSIX 兼容的方式来访问系统数据。在最新的 GlusterFS 3. 1. X 版本中，客户端不再需要独立维护卷配置信息，改成自动从运行在网关上的 Glusterd 弹性卷管理服务中获取和更新，极大简化了卷管理。GlusterFS 客户端的负载相对传统分布式文件系统要高，包括 CPU 占用率和内存占用率。

GlusterFS 存储网关提供弹性卷管理和 NFS/CIFS 访问代理功能，其上运行 Glusterd 和 Glusterfs 进程，两者都是 Glusterfsd 符号链接。卷管理器负责逻辑卷的创建、删除、容量扩展与缩减、容量平滑等功能，并负责向客户端提供逻辑卷信息及主动更新通知功能等。GlusterFS 3. 1. X 实现了逻辑卷的弹性和自动化管理，不需要中断数据服务或上层应用业务。对于 Windows 客户端或没有安装 GlusterFS 的客户端，需要通过 NFS/CIFS 代理网关来访问，这时网关被配置成 NFS 或 Samba 服务器。相对原生客户端，网关在性能上要受到 NFS/Samba 的制约。

GlusterFS 是模块化堆栈式的架构设计，如图 4-3 所示。模块称为 Translator，是 GlusterFS 提供的一种强大机制，借助这种良好定义的接口可以高效简便地扩展文件系统的功能。服务端与客户端模块接口是兼容的，同一个 Translator 可同时在两边加载。每个 Translator 都是 SO 动态库，运行时根据配置动态加载。每个模块实现特定基本功能，GlusterFS 中所有的功能都是通过 Translator 实现，如 Cluster（集群）、Storage（存储）、Performance（性能）、Protocol（协议）、Features（特点）等，基本简单的模块可以通过堆栈式的组合来实现复杂的功能。这一设计思想借鉴了 GNU/Hurd 微内核的虚拟文件系统设计，可以把对外部系统的访问转换成目标系统的适当调用。大部分模块都运行在客户端，如合成器、I/O 调度器和性能优化等，服务端相对简单许多。客户端和存储服务器均有自己的存储栈，构成了一棵 Translator 功能树，应用了若干模块。模块化和堆栈式的架构设计，极大降低了系统设计复杂性，简化了系统的实现、升级以及系统维护。

（4）弹性哈希算法

对于分布式系统而言，元数据的处理能力是决定系统扩展性、性能以及稳定性的关键。GlusterFS 另辟蹊径，彻底摒弃了元数据服务，使用弹性哈希算法代替传统分布式文件系统中的集中或分布式元数据服务。从根本上解决了元数据这一难题，从而获得了接近线性的高扩展性，同时也提高了系统的性能和可靠性。GlusterFS 使用算法进行数据定位，集群中的任何服务器和客户端只需根据路径和文件名就可以对数据进行定位和读写访问。换句话说，GlusterFS 不需要将元数据与数据进行分离，因为文件定位可独立并行化进行。GlusterFS 中数据访问流程如下：

① 计算 Hash 值，输入参数为文件路径和文件名。

② 根据 Hash 值在集群中选择子卷（存储服务器），进行文件定位。

③ 对所选择的子卷进行数据访问。

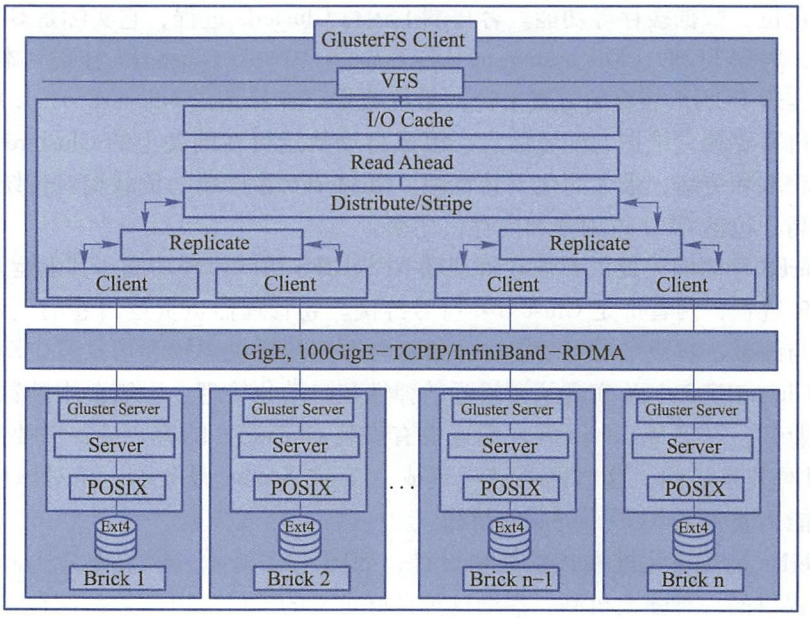


图 4-3 模块化堆栈设计

笔记

GlusterFS 目前使用 Davies-Meyer 算法计算文件名 Hash 值，获得一个 32 位整数。Davies-Meyer 算法具有非常好的 Hash 分布性，计算效率很高。假设逻辑卷中的存储服务器有 N 个，则 32 位整数空间被平均划分为 N 个连续子空间，每个空间分别映射到一个存储服务器。这样，计算得到的 32 位 Hash 值就会被投射到一个存储服务器，即要选择的子卷。现在读者考虑一下存储节点增加和删除、文件改名等情况，GlusterFS 如何解决这些问题而具备弹性的呢？

逻辑卷中增加一个新存储节点，如果不作其他任何处理，Hash 值映射空间将会发生变化，现有的文件目录可能会被重新定位到其他的存储服务器上，从而导致定位失败。解决问题的方法是对文件目录进行重新分布，把文件移动到正确的存储服务器上去，但这大大加重了系统负载，尤其是对于已经存储大量数据的海量存储系统来说显然是不可行的。另一种方法是使用一致性哈希算法，修改新增节点及相邻节点的 Hash 映射空间，仅需要移动相邻节点上的部分数据至新增节点，影响相对小了很多。然而，这又带来另外一个问题，即系统整体负载不均衡。GlusterFS 没有采用上述两种方法，而是设计了更为弹性的算法。GlusterFS 的哈希分布是以目录为基本单位的，文件的父目录利用扩展属性记录了子卷映射信息，其下面子目录在父目录所属存储服务器中进行分布。由于文件目录事先保存了分布信息，因此新增节点不会影响现有文件存储分布，它将从此后新创建的目录开始参与存储分布调度。这种设计，新增节点不需要移动任何文件，但是负载均衡没有平滑处理，老节点负载较重。GlusterFS 在设计中考虑了这一问题，在新建文件时会优先考虑容量负载最轻的节点，在目标存储节点上创建文件链接指向真正存储文件的节点。另外，GlusterFS 的弹性卷管理工具可以在后台以人工方式来执行负载均衡，将进行文件移动和重新分布，此后所有存储服务器都

会被调度。

GlusterFS 目前对存储节点删除支持有限，还无法做到完全无人干预的程度。如果直接删除节点，那么所在存储服务器上的文件将无法浏览和访问，创建文件目录也会失败。当前人工解决方法有两个，一是将节点上的数据重新复制到 GlusterFS 中，二是使用新的节点来替换删除节点并保持原有数据。

如果一个文件被改名，显然 Hash 算法将产生不同的值，可能会发生文件被定位到不同的存储服务器上，从而导致文件访问失败。采用数据移动的方法，对于大文件是很难实时完成的。为了不影响性能和服务中断，GlusterFS 采用了文件链接来解决文件重命名的问题，在目标存储服务器上创建一个链接指向实际的存储服务器，访问时由系统解析并进行重定向。另外，后台同时进行文件迁移，成功后文件链接将被自动删除。对于文件移动也作类似处理，好处是前台操作可实时处理，物理数据迁移置于后台选择适当时机执行。

弹性哈希算法为文件分配逻辑卷，那么 GlusterFS 如何为逻辑卷分配物理卷呢？GlusterFS 3. 1. X 实现了真正的弹性卷管理，存储卷是对底层硬件的抽象，可以根据需要进行扩容和缩减，以及在不同物理系统之间进行迁移。存储服务器可以在线增加和移除，并能在集群之间自动进行数据的负载均衡，数据总是在线可用，没有应用中断。文件系统配置更新也可以在线执行，所作配置变动能够快速动态地在集群中传播，从而自动适应负载波动和性能调优。

弹性哈希算法本身并没有提供数据容错功能，GlusterFS 使用镜像或复制来保证数据可用性，推荐使用镜像或 3 路复制。复制模式下，存储服务器使用同步写复制到其他的存储服务器，单个服务器故障完全对客户端透明。此外，GlusterFS 没有对复制数量进行限制，读被分散到所有的镜像存储节点，可以提高读性能。弹性哈希算法分配文件到唯一的逻辑卷，而复制可以保证数据至少保存在两个不同存储节点，两者结合使得 GlusterFS 具备更高的弹性。

任务实施

（1）环境准备

系统版本：CentOS-6. 5-x86_64-bin_DVD. iso。

系统节点见表 4-1。

表 4-1　系 统 节 点

IP 地址	主机名	挂载路径
10. 0. 6. 4	GlusterFS1	/export/brick1/gv0
10. 0. 6. 39	GlusterFS2	/export/brick1/gv0

（2）GlusterFS 安装配置

① 安装 GlusterFS 软件包。

配置 YUM 源，在/etc/yum. repos. d/目录下，添加一个 local. repo 文件。

```
[root@GlusterFS1 yum.repos.d]# vi local.repo
[glusterfs]
name=glusterfs
baseurl=https://download.gluster.org/pub/gluster/glusterfs/3.4/3.4.7/CentOS/epel-6.5/x86_64/
gpgcheck=0
enabled=1
[root@GlusterFS1 ~]# yum clean all
[root@GlusterFS1 ~]# yum list
```

配置 DNS 服务。

```
[root@GlusterFS1 ~]#vi /etc/resolv.conf
nameserver 114.114.114.114
```

在两个节点中安装 GlusterFS 需要的包。

GlusterFS 1 节点：

```
[root@GlusterFS1 ~]# yum install -y glusterfs-server xfsprogs
```

GlusterFS 2 节点：

```
[root@GlusterFS2 ~]# yum install -y glusterfs-server xfsprogs
```

安装完之后，启动服务并设置开机启动。

```
[root@GlusterFS1 ~]# service glusterd start
[root@GlusterFS1 ~]# chkconfig glusterd on
```

添加节点到 GlusterFS 集群。

```
[root@GlusterFS1 ~]# gluster peer probe 10.0.6.4
peer probe:success:on localhost not needed
[root@GlusterFS1 ~]# gluster peer probe 10.0.6.39
peer probe:success
```

② 查询状态。

查看各个节点的状态。

```
[root@GlusterFS1 ~]# gluster peer status
Number of Peers:1

Hostname:10.0.6.39
Port:24007
Uuid:8243e61b-330f-47fb-b352-ad70ae212f57
State:Peer in Cluster (Connected)
```

③ 创建目录。

创建数据存储目录（两个节点都要执行）。

先使用 Fdisk 分区工具将硬盘分出一个 10 GB 的分区。然后使用“lsblk”命令查看。

```
[root@GlusterFS1 ~]# fdisk  /dev/vda
[root@GlusterFS1 ~]# lsblk
NAME                          MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
vda                           252:0    0   50G  0 disk
├─vda1                        252:1    0  500M  0 part  /boot
├─vda2                        252:2    0  7.5G  0 part
│ ├─VolGroup-lv_root (dm-0)   253:0    0  6.7G  0 lvm   /
│ └─VolGroup-lv_swap (dm-1)   253:1    0  816M  0 lvm   [SWAP]
└─vda3                        252:3    0   10G  0 part
```

使用 XFS 文件系统对分区进行格式化。

```
[root@GlusterFS1 ~]# mkfs.xfs /dev/vda3
meta-data=/dev/vda3              isize=256    agcount=4, agsize=655389 blks
         =                       sectsz=512   attr=2, projid32bit=0
data     =                       bsize=4096   blocks=2621556, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

创建挂载目录。

```
[root@GlusterFS1 ~]# mkdir-p /export/brick1
```

挂载分区。

```
[root@GlusterFS1 ~]# mount /dev/vda3 /export/brick1/
```

查看挂载情况。

```
[root@GlusterFS1 ~]# df-h
Filesystem                    Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup-lv_root  6.7G  793M  5.5G  13% /
tmpfs                         939M     0  939M   0% /dev/shm
/dev/vda1                     485M   32M  428M   7% /boot
/dev/vda3                      10G   33M   10G   1% /export/brick1
```

创建存储目录。

```
[root@GlusterFS1 ~]# mkdir /export/brick1/gv0
```

GlusterFS 2 节点重复上述的操作，分区、格式化、挂载、创建存储目录。

④ 创建磁盘卷。

创建 GlusterFS 磁盘卷。

创建系统卷 gv 0（副本卷）。

```
[root@GlusterFS1 ~]# gluster volume create gv0 replica 2 10.0.6.4:/export/brick1/gv0 10.0.6.39:/export/brick1/gv0
```

启动系统卷 gv 0。

```
[root@GlusterFS1 ~]# gluster volume  start gv0
volume start:gv0:success
```

查看系统卷信息。

```
[root@GlusterFS1 ~]# gluster volume info

Volume Name:gv0
Type:Replicate
Volume ID:feb37096-fb17-4525-b033-07a731c796fb
Status:Started
Number of Bricks:1 x 2=2
Transport-type:tcp
Bricks:
Brick1:10.0.6.4:/export/brick1/gv0
Brick2:10.0.6.39:/export/brick1/gv0
```

笔记

⑤ 挂载文件系统。

安装客户端并挂载 GlusterFS 文件系统，使用 GlusterFS 2 节点作为客户端，在客户端挂载 GlusterFS 文件系统。

```
[root@GlusterFS2 ~]# mount -t glusterfs 10.0.6.4:/gv0 /mnt/
[root@GlusterFS2 ~]# df -h
Filesystem                  Size  Used  Avail  Use%  Mounted on
/dev/mapper/VolGroup-lv_root  6.7G  1.1G  5.3G  17%  /
tmpfs                       939M  0     939M   0%   /dev/shm
/dev/vda1                   485M  32M   428M   7%   /boot
/dev/vda3                   10G   33M   10G    1%   /export/brick1
10.0.6.4:/gv0               10G   33M   10G    1%   /mnt
```

验证成功，副本卷 gv 0 的大小是 10 GB，因为 GlusterFS 的副本数为 2，存储空间

有一半冗余。

（3）运维操作

添加节点（将节点 server ip 添加到存储池中）。

```
# gluster peer probe server ip
```

删除节点。

```
# gluster peers detach server ip
```

注意：将节点 server 从存储池中移除，移除节点是要保证节点上没有 Brick。如果节点上有 Brick，需要提前移除 Brick。

查看卷信息。

```
# gluster volume info
```

查看卷状态。

```
# gluster volume status
```

启动，停止卷。

```
# gluster volume start/stop VOLUME
```

删除卷。

```
# gluster volume delete VOLUME
```

修复卷。

```
# gluster volume heal mamm-volume#只修复有问题的文件
# gluster volume heal mamm-volume full #修复所有文件
# gluster volume heal mamm-volume info #查看自愈详情
```

（4）Brick 管理

添加 Brick。

```
#gluster peer probe 10.0.6.41
#gluster peer probe 10.0.6.42
#gluster volume add-brick gv0 10.0.6.41:/export/brick1/gv0 10.0.6.42:/export/brick1/gv0
```

注意：添加两个 Brick 到 gv 0，副本卷则要一次添加的 Bricks 数是 Replica 的整数倍，Stripe 同样要求。

移除 Brick。

```
#gluster volume remove-brick gv0 10.0.6.41:/export/brick1/gv0 10.0.6.42:/export/brick1/gv0 start
```

注意：若是副本卷，则要移除的 Brick 是 Replica 的整数倍，Stripe 具有同样的要求，副本卷要移除一对 Brick，在执行移除操作时，数据会移到其他节点。

在执行移除操作后，可以使用 status 命令对 task 状态进行查看。

```
#gluster volume remove-brick gv0 10.0.6.41:/export/brick1/gv010.0.6.42:/export/brick1/gv0 status
```

使用“commit”命令执行 Brick 移除，则不会进行数据迁移而直接删除 Brick，符合不需要数据迁移的用户需求。

```
# gluster volume remove - brick gv0 10.0.6.41:/export/brick1/gv0 10.0.6.42:/export/brick1/gv0 commit
```

项目实训

【实训题目】

使用两台 CentOS 6.5 虚拟机搭建一个 GlusterFS 文件系统并挂载使用。

【实训目的】

1. 掌握 GlusterFS 文件系统的搭建和使用。
2. 掌握 GlusterFS 文件系统的运维操作。

【实训内容】

1. 安装 GlusterFS 所需要的包，并开启服务。
2. 添加节点到 GlusterFS 集群。
3. 创建数据存储目录。
4. 创建 GlusterFS 磁盘卷。
5. 安装客户端并挂载 GlusterFS 磁盘卷，使用和管理 GlusterFS 磁盘卷。

单元小结

本单元主要学习了 GlusterFS 分布式文件系统的构建和运维。通过对本单元的学习，读者对 GlusterFS 文件系统的架构、部署、使用和运维应有一定的认识。GlusterFS 文件系统主要应用在集群系统中，具有很好的可拓展性。GlusterFS 解决了网络存储、联合存储（融合多个节点上的存储空间）、冗余备份等问题。但在实际的使用中，该文件系统的可靠性还需进一步验证。

单元 5

构建Ceph分布式存储系统

学习目标

【知识目标】

- 了解 Ceph 文件系统的基本组成。
- 熟悉 Ceph 分布式存储集群的架构。
- 掌握 Ceph 分布式存储集群的部署原理。

【技能目标】

- 掌握 Ceph 分布式存储集群的部署方法。
- 掌握 Ceph 分布式存储集群的基本运维。
- 掌握 Ceph 分布式存储集群的备灾处理方法。

PPT5：构建 Ceph 分布式存储系统

学习情境

研发部根据小缪的方案采用了 GlusterFS 文件系统后，还想再尝试下其他的文件系统，进行对比测试，找出性能比较好的一种方式，并集成到公司的私有云中。公司决定安排小缪测试下 Ceph 文件系统。

（1）项目设计

使用 3 台服务器搭建 1 个 Ceph 文件系统，并挂载使用。

（2）服务器功能

① CentOS 实现 Ceph 集群的搭建和使用。

② 实现基于 Ceph 文件系统的集群存储。

任务 5 构建 Ceph 分布式存储系统

任务描述

1. 了解 Ceph 文件系统的构架和使用环境。
2. 使用 3 台服务器搭建 1 个 Ceph 文件系统。
3. 实现基于 Ceph 文件系统的集群存储。

知识学习

笔 记

1. 云硬盘介绍

云硬盘是 IaaS 云平台的重要组成部分，云硬盘给虚拟机提供了持久的块存储设备。目前的 AWS 的 EBS（Elastic Block Store）给亚马逊的 EC2 实例提供了高可用高可靠的块级存储卷，EBS 适合于一些需要访问块设备的应用，如数据库、文件系统等。在 OpenStack 中，可以使用 Ceph、Sheepdog、GlusterFS 作为云硬盘的开源解决方案，以下是 Ceph 的架构特点。

首先，Ceph 是统一存储系统，支持如下 3 种接口。

① Object：有原生的 API，而且也兼容 Swift 和 S3 的 API。

② Block：支持精简配置、快照和克隆。

③ File：POSIX 接口，支持快照。

其次，Ceph 也是分布式存储系统，它具有如下特点。

① 高扩展性：使用普通 x86 服务器，支持 10~1,000 台服务器，支持 TB 到 PB 级的扩展。

② 高可靠性：没有单点故障，拥有多数据副本、自动管理和自动修复功能。

③ 高性能：数据分布均衡，并行化度高。对于对象存储（Objects Storage）和块存储（Block Storage），不需要元数据服务器。

2. Ceph 背景

目前红帽（Red Hat）公司掌控 Ceph 的开发，但 Ceph 是开源的，遵循 LGPL（GNU Lesser General Public License，GNU 宽通用公共许可证）协议。红帽公司还积极整合 Ceph 配合其他的云计算和大数据平台，目前 Ceph 支持 OpenStack、CloudStack、OpenNebula、Hadoop 等。

当前 Ceph 的最新稳定版本为 0.67（Dumpling）版本，它的对象存储和块存储已经足够稳定，而且 Ceph 社区还在继续开发新功能，包括跨机房部署、容灾和支持 Erasure Encoding 等。Ceph 具有完善的社区设施和发布流程（每 3 个月发布 1 个稳定版本）。

目前 Ceph 有很多用户案例，从调查中可以看到有 26%的用户在生产环境中使用

Ceph，有 37%的用户在私有云中使用 Ceph，还有 16%的用户在公有云中使用 Ceph。

目前 Ceph 最大的用户案例是 Dreamhost 的 Object Service（对象服务），目前总容量是 3 PB，可靠性达到 99.99999%。数据存放采用 3 个副本，它的价格比 S3 还便宜。

3. Ceph 架构

Ceph 构架如图 5-1 所示。

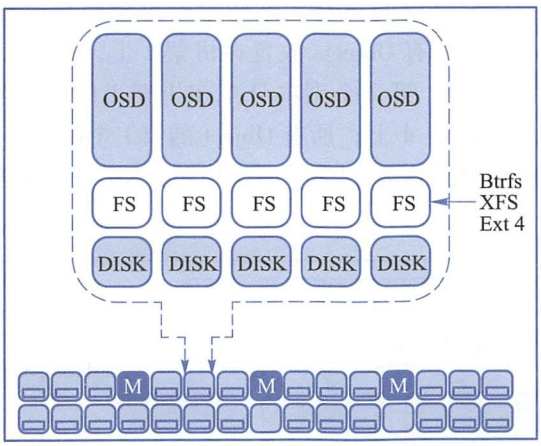


图 5-1　Cpeh 架构图

（1）RADOS 组成

Ceph 的底层是 RADOS（reliable autonomous distributed object storage，可靠的、自主的、分布式对象存储系统）。RADOS 由以下两个组件组成。

① OSD：Object Storage Device，提供存储资源。

② Monitor：维护整个 Ceph 集群的全局状态。

RADOS 具有很强的可扩展性和可编程性，Ceph 基于 RADOS 开发了 Object Storage（对象存储）、Block Storage（块存储）和 FileSystem（文件系统）。

（2）Ceph 组件

① MDS：用于保存 CephFS 的元数据。

② RADOS Gateway：对外提供 REST 接口，兼容 S3 和 Swift 的 API。

（3）Ceph 映射

Ceph 的命名空间是（Pool，Object），每个 Object 都会映射到一组 OSD 中，由这组 OSD 保存这个 Object，其映射关系如下：

（Pool，Object）→（Pool，PG）→OSD SET→Disk

Ceph 中 Pools 的属性包括 Object 的副本数、Placement Groups（数据映射）的数量、所使用的 CRUSH Ruleset。

在 Ceph 中，Object 先映射到 PG（Placement Group），再由 PG 映射到 OSD SET。每个 Pool 有多个 PG，每个 Object 通过计算 Hash 值并得到它所对应的 PG。PG 再映射到一组 OSD（OSD 的个数由 Pool 的副本数决定），第一个 OSD 是 Primary，剩下的都

笔记

是 Replicas。

数据映射的方式决定了存储系统的性能和扩展性。（Pool，PG）→OSD SET 的映射由以下 4 个因素决定。

① CRUSH 算法：一种伪随机算法。

② OSD MAP：包含当前所有 Pool 的状态和所有 OSD 的状态。

③ CRUSH MAP：包含当前磁盘、服务器、机架的层级结构。

④ CRUSH Rules：数据映射的策略。这些策略可以灵活地设置 Object 存放的区域，例如可以指定 Pool 1 中所有 Objects 放置在机架 1 上，所有 Objects 的第 1 个副本放置在机架 1 的服务器 A 上，第 2 个副本分布在机架 1 的服务器 B 上。Pool 2 中所有的 Object 分布在机架 2、3、4 上，所有 Object 的第 1 个副本分布在机架 2 的服务器上，第 2 个副本分布在机架 3 的服务器上，第 3 个副本分布在机架 4 的服务器上，如图 5-2 所示。

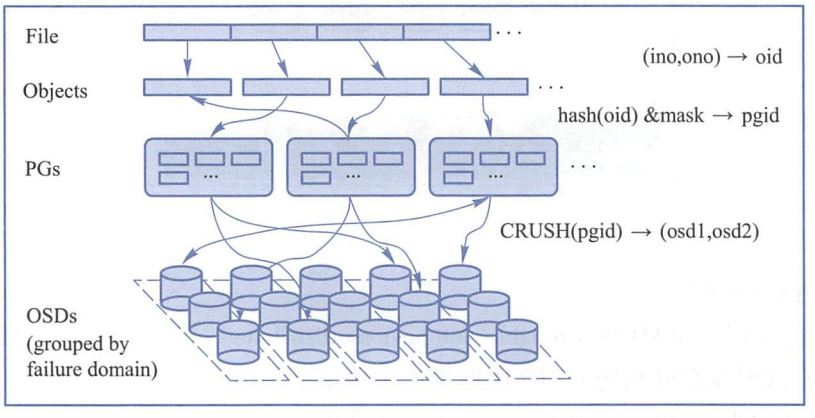


图 5-2 数据映射的策略

Client 从 Monitors 中得到 CRUSH MAP、OSD MAP、CRUSH Ruleset，然后使用 CRUSH 算法计算出 Object 所在的 OSD SET。所以 Ceph 不需要 Name 服务器，Client 直接和 OSD 进行通信。其伪代码如下：

```
locator = object_name
obj_hash = hash(locator)
pg = obj_hash % num_pg
osds_for_pg = crush(pg)
primary = osds_for_pg[0]
replicas = osds_for_pg[1:]
```

（4）数据映射的优点

① 把 Object 分组，这降低了需要追踪和处理的 Metadata（元数据）数量（在全局的层面上，不需要追踪和处理每个 Object 的 Metadata 和 Placement，只需要管理 PG 的 Metadata 就可以了。PG 的数量级远远低于 Object 的数量级）。

② 增加 PG 的数量，可以均衡每个 OSD 的负载，提高并行度。

③ 分隔故障域，提高数据的可靠性。

④ 强一致性。Ceph 的读写操作采用 Primary-Replica 模型，Client 只向 Object 所对应 OSD SET 的 Primary 发起读写请求，这保证了数据的强一致性。由于每个 Object 都只有一个 Primary OSD，因此对 Object 的更新都是顺序的，不存在同步问题。当 Primary 收到 Object 的写请求时，它负责把数据发送给其他 Replicas，只要这个数据被保存在所有的 OSD 上时，Primary 才应答 Object 的写请求，这保证了副本的一致性，如图 5-3 所示。

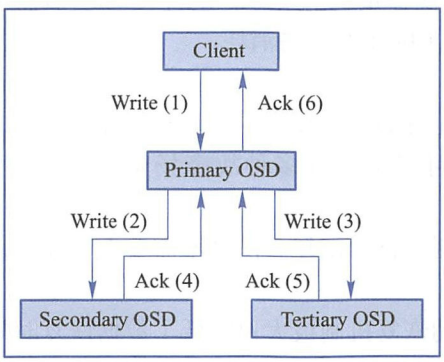


图 5-3　副本一致性

⑤ 容错性。在分布式系统中，常见的故障有网络中断、掉电、服务器宕机、硬盘故障等，Ceph 能够容忍这些故障，并进行自动修复，保证数据的可靠性和系统可用性。Monitors 是 Ceph 管家，维护着 Ceph 的全局状态。Monitors 的功能和 ZooKeeper（分布式系统的可靠协调系统）类似，它们使用 Quorum 和 Paxos 算法去建立全局状态的共识。OSDs 可以进行自动修复，而且是并行修复。

笔记

（5）故障检测

OSD 之间有心跳检测，当 OSD A 检测到 OSD B 没有回应时，会报告给 Monitors 说 OSD B 无法连接，则 Monitors 给 OSD B 标记为 down 状态，并更新 OSD Map。当过了 M 秒（可以在 Ceph 中配置 M 的值）之后还是无法连接到 OSD B，则 Monitors 给 OSD B 标记为 out 状态（表明 OSD B 不能工作），并更新 OSD Map。

（6）故障恢复

当某个 PG 对应的 OSD SET 中有一个 OSD 被标记为 down 时（如果 Primary 被标记为 down，则某个 Replica 会成为新的 Primary，并处理所有读写 Object 请求），则该 PG 处于 active+degraded 状态，也就是当前 PG 有效的副本数是 N-1。

过了 M 秒之后，假如还是无法连接该 OSD，则它被标记为 out，Ceph 会重新计算 PG 到 OSD SET 的映射（当有新的 OSD 加入到集群时，也会重新计算所有 PG 到 OSD SET 的映射），以此保证 PG 的有效副本数是 N。

新 OSD SET 的 Primary 先从旧的 OSD SET 中收集 PG log，得到一份 Authoritative History（完整的、全序的操作序列），并让其他 Replicas 同意这份 Authoritative History

（也就是其他 Replicas 对 PG 的所有 Objects 状态都达成一致），这个过程叫作 Peering。

当 Peering 过程完成之后，PG 进入 active+recoverying 状态，Primary 会迁移和同步那些降级的 Objects 到所有的 Replicas 上，保证这些 Objects 的副本数为 N。

笔 记

4. Ceph 的优点

（1）高性能

Client 和 Server 直接通信，不需要代理和转发。同时是由多个 OSD 带来的高并发度，而 Objects 是分布在所有 OSD 上的，所以一定程度上可以进行负载均衡，每个 OSD 都有权重值（现在以容量为权重）。这样 Client 不需要负责副本的复制（由 Primary 负责），这降低了 Client 的网络消耗。

（2）高可靠性

① 数据多副本。可配置 per-pool 副本策略和故障域布局，支持强一致性。

② 没有单点故障。可以忍受许多种故障场景，单个组件可以滚动升级并在线替换。

③ 所有故障自动检测和自动恢复。恢复不需要人工介入，在恢复期间，可以保持正常的数据访问。

④ 并行恢复。并行的恢复机制极大地降低了数据恢复时间，提高数据的可靠性。

（3）高扩展性

① 高度并行。没有单个中心控制组件。所有负载都能动态的划分到各个服务器上。把更多的功能放到 OSD 上，让 OSD 更智能。

② 自管理。容易扩展、升级和替换。当组件发生故障时，自动进行数据的重新复制。当组件发生变化时（添加或删除），自动进行数据的重新分布。

在单机情况下，RBD 的性能不如传统的 RAID 10，这是因为 RBD 的 I/O 路径很复杂，导致效率很低。但是 Ceph 的优势在于它的扩展性，它的性能会随着磁盘数量线性增长，因此在多服务器的情况下，RBD 的 IOPS 和吞吐率会高于单机的 RAID 10（不过性能会受限于网络的带宽）。

综上所述，Ceph 优势显著，使用它能够降低硬件成本和运维成本，但其复杂性会带来一定的成本。

任务实施

1. 基础环境配置

① 创建 3 个 CentOS 7 虚拟机，分别命名为 ceph-node1、ceph-node2 和 ceph-node3。配置 3 台虚拟机的 IP 分别为 192.168.1.101、192.168.1.102 和 192.168.1.103，为了方便起见，修改虚拟机的/etc/hosts 文件。

```
192.168.1.101   ceph-node1
192.168.1.102   ceph-node2
192.168.1.103   ceph-node3
```

② 在 ceph-node1 节点生成 Root SSH 密钥，并将它复制到 ceph-node2 和 ceph-node3 上。这些虚拟机的 root（用户）的密码都是 000000。在使用“ssh-copy-id”命令时如果要输入该密码。

```
# ssh-keygen
# ssh-copy-id root@ ceph-node2
# ssh-copy-id root@ ceph-node3
```

③ 复制密钥到 ceph-node2 和 ceph-node3 之后，ceph-node1 就可以无密钥登录到另外两台虚拟机上了。

④ 关闭防火墙。

```
# service firewalld stop
```

⑤ 禁用 3 台虚拟机上的 SELinux。

临时禁用。

```
# setenforce 0
```

永久禁用。

```
# vi /etc/selinux/config 把 selinux 改成 disablesd
```

⑥ 在所有虚拟机上安装并配置 NTP 服务和网络的时钟服务器同步时间。

```
# yum install -y ntp ntpdate
# ntpdate pool. ntp. org
# chkconfig ntpd on
```

⑦ 在所有 Ceph 节点上修改 YUM 源，均使用本地 FTP 服务器源（软件包参看随书资源）。

```
# cat /etc/yum. repos. d/local. repo
[centos7. 2]
name=centos7. 2
baseurl=ftp://10. 0. 0. 254/file/cr/2017/centos7. 2/
gpgcheck=0
enabled=1
[iaas]
name=iaas
baseurl=ftp://10. 0. 0. 254/file/cr/2017/train/iaas-repo/
gpgcheck=0
enabled=1
```

2. 安装和配置 Ceph

要部署这个集群，需要使用 ceph-deploy 工具在 3 台虚拟机上安装和配置 Ceph。ceph-deploy 是 Ceph 软件定义存储系统的一部分，用来方便地配置和管理 Ceph 存储集群。

笔记

(1) 创建 Ceph 集群

首先，在 ceph-node1 上安装 Ceph，并配置它为 Ceph monitor 和 OSD 节点。

① 在 ceph-node1 上安装 ceph-deploy。

```
[root@ceph-node1 ~]# yum install ceph-deploy -y
```

② 通过在 cep-node1 上执行以下命令，用 ceph-deploy 创建一个 Ceph 集群。

```
[root@ceph-node1 ~]# mkdir /etc/ceph
[root@ceph-node1 ~]# cd /etc/ceph
[root@ceph-node1 ceph]# ceph-deploy new ceph-node1
```

③ ceph-deploy 的 new 子命令能够部署一个默认名称为 Ceph 的新集群，并且它能生成集群配置文件和密钥文件。列出当前的工作目录，可以查看到 ceph. conf 和 ceph. mon. keying 文件。

```
[root@ceph-node1 ceph]# ceph-deploy new ceph-node1
[ceph_deploy.conf][DEBUG ] found configuration file at:/root/.cephdeploy.conf
[ceph_deploy.cli][INFO  ] Invoked(1.5.36):/usr/bin/ceph-deploy new ceph-node2
[ceph_deploy.new][DEBUG ] Creating new cluster named ceph
[ceph_deploy.new][INFO  ] making sure passwordless SSH succeeds
[ceph-node2][DEBUG ] connected to host:ceph-node2
[ceph-node2][INFO  ] Running command:/usr/sbin/ip addr show
[ceph_deploy.new][DEBUG ] Writing initial config to ceph.conf...
```

④ 在 ceph-node1 上执行以下命令，使用 ceph-deploy 工具在所有节点上安装 Ceph 二进制软件包。

```
[root@ceph-node1 ceph]# ceph-deploy install ceph-node1 ceph-node2 ceph-node3
[ceph_deploy.conf][DEBUG ] found configuration file at:/root/.cephdeploy.conf
 [ceph-node1][DEBUG ] connected to host:ceph-node1
[ceph-node1][DEBUG ] detect platform information from remote host
[ceph-node1][DEBUG ] detect machinetype
...
[ceph-node3][DEBUG ] Upgrade  2 Packages(+14 Dependent packages)
[ceph-node3][DEBUG ]
```

```
[ceph-node3][DEBUG ] Total download size:47 M
[ceph-node3][DEBUG ] Downloading packages:
[ceph-node3][DEBUG ] Delta RPMs disabled because/usr/bin/applydeltarpm not installed.
[ceph-node3][WARNIN] No data was received after 300 seconds,disconnecting...
[ceph-node3][INFO  ] Running command:ceph --version
[ceph-node3][DEBUG ] ceph version 10.2.3(ecc23778eb545d8dd55e2e4735b53cc93f92e65b)
```

⑤ ceph-deploy 工具包首先会安装 Ceph 组件所有的依赖包。命令成功完成后，检查所有节点上 Ceph 的版本信息。

```
[root@ceph-node1 ceph]# ceph -v
ceph version 0.94.5(9764da52395923e0b32908d83a9f7304401fee43)
```

⑥ 在 ceph-node1 上创建第一个 Ceph monitor。

```
[root@ceph-node1 ceph]# ceph-deploy --overwrite-conf mon create-initial
[ceph_deploy.conf][DEBUG ] found configuration file at:/root/.cephdeploy.conf
[ceph_deploy.cli][INFO  ] Invoked(1.5.36):/usr/bin/ceph-deploy --overwrite-conf mon create-initial
[ceph-node1][DEBUG ] write cluster configuration to /etc/ceph/{cluster}.conf
[ceph-node1][DEBUG ] create the mon path if it does not exist
[ceph-node1][DEBUG ] checking for done path:/var/lib/ceph/mon/ceph-ceph-node1/done
[ceph-node1][DEBUG ] done path does not exist:/var/lib/ceph/mon/ceph-ceph-node1/done
[ceph-node1][INFO  ] Running command:systemctl enable ceph-mon@ceph-node1
[ceph_deploy.gatherkeys][INFO  ] Storing ceph.bootstrap-rgw.keyring
[ceph_deploy.gatherkeys][INFO  ] Destroy temp directory /tmp/tmpaTvGOj
```

⑦ Monitor 创建成功后，检查集群的状态，此时 Ceph 集群并不处于健康状态。

```
# ceph -s
    cluster ce58f331-1699-4c82-a003-196a354a73f2
     health HEALTH_ERR
            no osds
     monmap e1:1 mons at {ceph-node1=10.0.0.4:6789/0}
            election epoch 3,quorum 0 ceph-node1
     osdmap e1:0 osds:0 up,0 in
```

```
            flags sortbitwise,require_jewel_osds
      pgmap v2:64 pgs,1 pools,0 bytes data,0 objects
            0 kB used,0 kB / 0 kB avail
                  64 creating
```

（2）创建 OSD

① 列出 ceph-node1 上所有的可用磁盘。

```
[root@ceph-node1 ceph]# ceph-deploy disk list ceph-node1
[ceph_deploy.conf][DEBUG ] found configuration file at:/root/.cephdeploy.conf
[ceph_deploy.cli][INFO  ] Invoked(1.5.36):/usr/bin/ceph-deploy disk list ceph
-node1
[ceph_deploy.cli][INFO  ]ceph-deploy options:
[ceph_deploy.cli][INFO  ]  username          :None
[ceph_deploy.cli][INFO  ]  verbose           :False
[ceph_deploy.cli][INFO  ]  overwrite_conf    :False
[ceph_deploy.cli][INFO  ]  subcommand        :list
[ceph_deploy.cli][INFO  ]  quiet             :False
[ceph_deploy.cli][INFO  ]  cd_conf           :<ceph_deploy.conf.cephdeploy.
Conf instance at 0x7f3a3ee9d6c8>
[ceph_deploy.cli][INFO  ]  cluster           :ceph
[ceph_deploy.cli][INFO  ]  func              :<function disk at 0x7f3a3ee8fb90>
[ceph_deploy.cli][INFO  ]  ceph_conf         :None
[ceph_deploy.cli][INFO  ]  default_release   :False
[ceph_deploy.cli][INFO  ]  disk              :[('ceph-node1',None,None)]
[ceph-node1][DEBUG ] connected to host:ceph-node1
[ceph-node1][DEBUG ] detect platform information from remote host
[ceph-node1][DEBUG ] detect machine type
[ceph-node1][DEBUG ] find the location of an executable
[ceph_deploy.osd][INFO  ] Distro info:CentOS Linux 7.3.1611 Core
[ceph_deploy.osd][DEBUG ] Listing disks on ceph-node1...
[ceph-node1][DEBUG ] find the location of an executable
[ceph-node1][INFO  ] Running command:/usr/sbin/ceph-disk list
[ceph-node1][DEBUG ] /dev/vda :
[ceph-node1][DEBUG ]  /dev/vda1 other,xfs,mounted on /
```

② 创建共享磁盘，3 个节点都要执行。

```
[root@ceph-node1 ceph]#mkdir -p /opt/osd1
# chmod 777 /opt/osd1
```

```
[root@ceph-node1 ceph]# ssh ceph-node2
# mkdir -p /opt/osd2
# chomd777 /opt/osd2/
[root@ceph-node1 ceph]# ssh ceph-node3
# mkdir -p /opt/osd3
# chomd777 /opt/osd3/
```

③ 在 node1 节点使用 ceph-deploy 工具创建 OSD 节点。

```
[root@ceph-node1ceph]# ceph-deployosd prepare ceph-node1:/opt/osd1 ceph-node2:/opt/osd2 ceph-node3:/opt/osd3
```

④ 在 node1 节点使用 ceph-deploy 工具激活 OSD 节点。在激活 OSD 节点前，把创建的 osd 目录中的文件权限改为 777。

笔记

```
[root@ceph-node1ceph]#ceph-deploy osd activate ceph-node1:/opt/osd1 ceph-node2:/opt/osd2 ceph-node3:/opt/osd3
        [ceph_deploy.conf][DEBUG ] found configuration file at:/root/.cephdeploy.conf
        [ceph_deploy.cli][INFO  ]  cluster                    :ceph
        [ceph-node1][WARNIN] DEBUG:ceph-disk:OSD uuid is 6c5cd76e-f68c-48b2-b4b9-97d550970e65
        [ceph-node1][WARNIN] DEBUG:ceph-disk:Allocating OSD id...
        ...
        [ceph-node2][WARNIN] DEBUG:ceph-disk:Cluster uuid is 06b91a6e-8faf-4992-b95c-84d192f30096
        ...
        [ceph-node3][WARNIN] DEBUG:ceph-disk:OSD uuid is 70b154fa-fa5e-4e78-9d83-bcc18e09b9b6
        [ceph-node3][WARNIN]      a requirement dependency on it.
        [ceph-node3][WARNIN] 3) A unit may be started when needed via activation (socket,path,timer,
        [ceph-node3][WARNIN]      D-Bus,udev,scripted systemctl call,...).
```

⑤ 检查 Ceph 集群的状态。此时，集群是 HEALTH_OK 状态。

```
# ceph -s
    cluster 06b91a6e-8faf-4992-b95c-84d192f30096
     health HEALTH_OK
     monmap e1:1 mons at {ceph-node1=10.0.0.4:6789/0}
            election epoch 2,quorum 0 ceph-node1
     osdmap e15:3 osds:3 up,3 in
```

```
pgmap v40:64 pgs,1 pools,0 bytes data,0 objects
18722 MB used,101 GB / 119 GB avail
64 active+clean
```

⑥ 开放权限给其他节点，进行灾备处理。

```
# ceph-deploy admin ceph-node{1,2,3}
[ceph_deploy.conf][DEBUG ] found configuration file at:/root/.cephdeploy.conf
[ceph_deploy.cli][INFO  ] Invoked(1.5.31):/usr/bin/ceph-deploy admin ceph-node1 ceph-node2 ceph-node3
[ceph_deploy.admin][DEBUG ] Pushing admin keys and conf to ceph-node1
[ceph_deploy.admin][DEBUG ] Pushing admin keys and conf to ceph-node2
[ceph-node2][DEBUG ] connected to host:ceph-node2
[ceph-node2][DEBUG ] detect platform information from remote host
[ceph_deploy.admin][DEBUG ] Pushing admin keys and conf to ceph-node3
[ceph-node3][DEBUG ] connected to host:ceph-node3
[ceph-node3][DEBUG ] write cluster configuration to /etc/ceph/{cluster}.conf
# chmod +r /etc/ceph/ceph.client.admin.keyring
```

（3）Ceph 集群运维

有了可运行的 Ceph 集群后，现在可以用一些简单的命令来体验 Ceph。

① 检查 Ceph 的安装状态。

```
# ceph status
```

② 观察集群的健康状况。

```
# ceph -w
```

③ 检查 Ceph monitor 仲裁状态。

```
# ceph quorum_status --format json-petty
```

④ 导出 Ceph monitor 信息。

```
# ceph mon dump
```

⑤ 检查集群使用状态。

```
# ceph df
```

⑥ 检查 Ceph Monitor、OSD 和 PG（配置组）状态。

```
# ceph mon stat
# ceph osd stat
# ceph pg stat
```

⑦ 列表 PG。

```
# ceph pg dump
```

⑧ 列表 Ceph 存储池。

```
# ceph osd lspools
```

⑨ 检查 OSD 的 CRUSH。

```
# ceph osd tree
```

⑩ 列表集群的认证密钥。

```
# ceph auth list
```

项目实训

【实训题目】

在 CentOS 7 上构建 Ceph 的分布式云存储平台。

【实训目的】

1. 掌握 Ceph 云存储文件系统的简单使用。
2. 掌握 Ceph 分布式存储的构建方法。
3. 掌握 Ceph 分布式存储集群的运维方法。

【实训内容】

1. 在各个节点配置节点的无秘钥登录配置。
2. 在各个节点配置 Ceph 的 YUM 源和 CentOS 的 YUM 源。
3. 在各个节点安装 Ceph 的服务包。
4. 创建 Ceph 分布式存储集群。
5. 在集群里加入 OSD 节点。

单元小结

通过在本单元对 Ceph 的分布式存储平台的学习，读者应了解 Ceph 文件系统的基本组成和使用，熟悉 Ceph 分布式存储集群的架构，掌握 Ceph 分布式存储集群的部署原理、部署方法及基本运维。Ceph 分布式存储集群在大型数据心中被越来越多地使用，技术实现简单易懂，优势越来越明显，在云存储的技术发展中发挥越来越重要的作用。

单元 6

构建超融合基础架构

学习目标

【知识目标】

- 了解 Ceph 客户端的基本部署方法。
- 了解 OpenStack 和 Ceph 对接的方法。

【技能目标】

- 掌握 Ceph 客户端的基本部署方法。
- 掌握 Ceph 与 OpenStack 服务的对接配置。

PPT6：构建超融合基础架构

学习情境

小缪在对比了 GlusterFS 文件系统和 Ceph 文件系统后，认为 Ceph 集群更适合公司的使用，因此决定使用 Ceph 集群替换现有的 OpenStack 的后端存储，使公司私有云的存储得到统一。

（1）项目设计

使用已经搭建完成的 Ceph 集群，对接公司私有云的各个服务。

（2）服务器功能实现

① 使用 Ceph 集群替换 OpenStack 的 Glance。

② 使用 Ceph 集群替换 OpenStack 的 Nova。

任务 6　配置 OpenStack 使用 Ceph 存储

任务描述

1. 配置 OpenStack 的 Ceph 客户端。
2. 配置 OpenStack Glance 服务对接 Ceph。
3. 配置 OpenStack Nova 服务对接 Ceph。

知识学习

笔 记

1. 使用 Ceph 块设备

原名 RADOS 块设备，提供可靠的分布式和高性能块存储磁盘给客户端。RADOS 块设备使用 Librbd 库，把一个块数据以顺序条带化的形式存放在 Ceph 集群的多个 OSD 上。RBD 是建立在 Ceph 的 RADOS 层之上的，因此，每一个块设备都会分布在多个 Ceph 节点上，以提供高性能和高可靠性。RBD 原生支持 Linux 内核，这意味着在过去几年中 RBD 驱动已经完美地集成在 Linux 内核中了。除了可靠性和性能，RBD 还提供了企业特性，如完整和增量快照、自动精简配置、写时复制克隆、动态调整大小等。RBD 还支持内存内缓存，从而大大提升了性能，如图 6-1 所示。

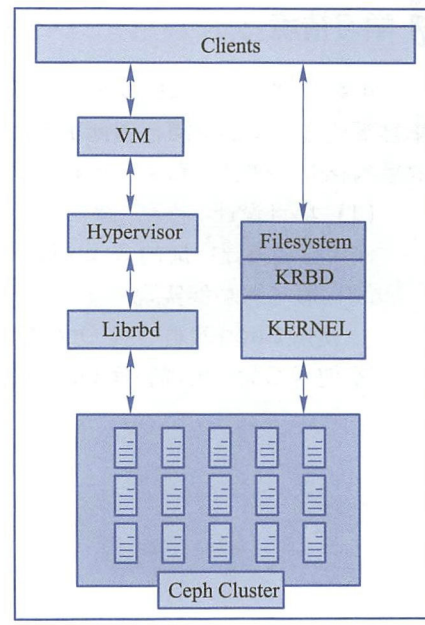


图 6-1　Ceph 集群

2. OpenStack 存储

OpenStack 项目中 Glance、Nova、Cinder、Swift 等服务是项目中存储的服务，服务

种类的不同也决定着提供服务的方式不同，在 OpenStack 的设计中 Glance 被用于镜像资源的存储，称为镜像服务。Nova 则用于虚拟机实例操作和实例资源的存储，称为计算服务。Cinder 则是提供虚拟机的块存储服务。Swift 则是提供对象存储服务，在下面的任务实施中重点讲解 Ceph 与 Glance、Nova 两者之间的结合和作为 OpenStack 服务后端统一存储的配置说明。

任务实施

1. 配置 OpenStack 作为 Ceph 客户端

（1）客户端无密钥登录

本任务将在 ceph-node1 上通过手工方式在 Training 节点（OpenStack 节点，这里部署的是 All_In_One 方式）上安装 Ceph 二进制程序，此时需要将 ssh 设置为无密钥登录 Training，Root（用户）密码同样为 000000。

```
# ssh-copy-id -i /root/. ssh/id_rsa. pub Training
```

（2）安装 Ceph

在 Training 节点上安装 Ceph。使用本书提供的 ceph-centos 6. 5 源。

```
#yum install ceph -y
```

（3）同步 Ceph 配置文件

将 Ceph 配置文件 ceph. conf 从 ceph-node1 推送到 Training。该配置文件会帮助客户端访问 Ceph Monitor 和 OSD 设备，开发者也可以手动将 ceph. conf 复制到 Training。

```
# cat /etc/ceph/ceph. conf  | ssh Training  tee /etc/ceph/ceph. conf
```

（4）配置存储池

为 Cinder、Glance、Nova 创建 Ceph 存储池。开发者也可以使用任何可用的存储池，这里使用默认的 RBD 作为 3 种存储的后端存储池，创建完成后可以检查当前的存储池信息。

```
# ceph osd pool stats
pool rbd id 0
  nothing is going on
```

（5）创建 Ceph 用户

为存储池创建认证用户。

```
# ceph auth get-or-create client. rbd mon 'allow r' osd 'allow class-read object_prefix rbd_children,allow rwx pool=rbd'
[client. rbd]
        key=AQDSXmhZReClMBAAmVO6ICrDA/Vr2e92vLo5ZQ==
```

（6）修改用户权限

创建 Training 节点的 keyring。

```
# ceph auth get-or-create client.rbd | ssh Training tee /etc/ceph/ceph.client.rbd.keyring
[client.rbd]
        key = AQDSXmhZReClMBAAmVO6ICrDA/Vr2e92vLo5ZQ==
```

（7）修改权限

修改 Training 节点的 keyring 权限。

```
# ssh Training chmod 777 /etc/ceph/ceph.client.rbd.keyring
# ceph auth get-key client.rbd | ssh Training tee client.rbd.key
```

（8）登录 Ceph 集群

现在，可以在 Training 上使用用户 client.rbd 来访问 Ceph 集群了。登录到 Training 并执行以下命令。

```
[root@Training ~]# ceph -s --name client.rbd --keyring /etc/ceph/ceph.client.rbd.keyring
    cluster 00b29894-03be-4b1b-bfe7-60e378167e63
     health HEALTH_OK
     monmap e1:1 mons at {ceph-node1=10.0.6.36:6789/0}
            election epoch 2,quorum 0 ceph-node1
     osdmap e27:3 osds:3 up,3 in
      pgmap v202:64 pgs,1 pools,0 bytes data,0 objects
            18692 MB used,581 GB / 599 GB avail
                  64 active+clean
```

（9）Ceph 客户端设备创建

到此为止，完成 Ceph 客户端的配置。以下是演示在 Training 节点上创建 Ceph 块设备的过程。

① 创建块设备。创建一个名称为 rbd 1，大小为 10,240 MB（10 GB）的 RADOS 块设备。

```
# rbd create rbd1 --size 10240 --name client.rbd
# rbd list --name client.rbd
rbd1
```

② 列举块设备。有多种选项帮助开发者列出 RBD 镜像，保存块设备镜像的默认存储池为 RBD，开发者也可以通过 RBD 的“-p”命令指定一个存储池。

```
# rbd ls --name client.rbd
rbd1
```

```
#   rbd ls -p rbd --name client. rbd
rbd1
# rbd list --name client. rbd
rbd1
```

③ 检查块设备。可以通过以下的命令检查 RBD 镜像的大小和其他信息。

```
# rbd --image rbd1 info --name client. rbd
rbd image 'rbd1':
        size 10240 MB in 2560 objects
        order 22(4096 kB objects)
        block_name_prefix:rb. 0. 1657. 74b0dc51
        format:1
```

至此，关于配置 OpenStack 作为 Ceph 客户端部分已经基本完成，接下来开始对每个服务配置使用 Ceph 存储。

笔记

（10）调整 Ceph RBD 大小

Ceph 支持精简配置的块设备，这意味着，直到用户开始在块设备上存储数据前，物理存储空间都不会被占用。Ceph 块设备非常灵活，开发者可以在 Ceph 存储端增加或减少 RBD 的大小。然而，底层的文件系统必须支持大小的调整。高级文件系统，如 XFS、Btrfs、Ext 4 和 ZFS 和其他文件系统都在一定程度上支持大小调整。

要增加或减少 Ceph RBD 镜像大小，使用 Rbd Resize 的“--size<New_Size_in_MB>”命令，它会设置 RBD 镜像新的大小。

例如，之前创建的 RBD 镜像的大小为 10 GB，现在把其增加到 20 GB。

```
# rbd resize --image rbd1 --size 20480 --name client. rbd
Resizing image:100% complete. . . done.
# rbd info --image rbd1 --name client. rbd
rbd image 'rbd1':
        size 20480 MB in 5120 objects
        order 22(4096 kB objects)
        block_name_prefix:rb. 0. 1657. 74b0dc51
        format:1
```

2. 配置 Glance 服务

现在已经完成了 Ceph 侧所需的配置，接下来通过配置 OpenStack Glance，将 Ceph 用作后端存储，配置 OpenStack Glance 模块来将其虚拟机镜像存储在 Ceph RDB 中。

（1）修改 Glance 配置文件

登录到 Training 节点，然后编辑/etc/glance/glance-api. conf 文件的［DEFAULT］下的配置文件并做如下修改。

```
# cat /etc/glance/glance-api.conf | grep -v ^# | grep -v ^$
[DEFAULT]
show_image_direct_url=True
default_store=rbd
filesystem_store_datadir=/var/lib/glance/images/
rbd_store_ceph_conf=/etc/ceph/ceph.conf
rbd_store_user=rbd
rbd_store_pool=rbd
rbd_store_chunk_size=8
```

（2）重新启动服务

重新启动 OpenStack Glance 服务。

```
# service openstack-nova-compute restart
```

（3）检查结果

① 转换镜像。要在 Ceph 中启动虚拟机，Glance 镜像的格式必须为 RAW。这里可以利用本书提供的 cirros-0.3.4-x86_64-disk.img 镜像，将镜像类型从 QCOW2 转换成 RAW 格式。也可以使用任何 RAW 格式的其他镜像。

```
# qemu-img convert -p -f qcow2 -O raw cirros-0.3.4-x86_64-disk.img cirros.raw
```

② 上传镜像。将修改的镜像上传到系统。

```
glance image-create --name="CirrOS0.3.4" --disk-format=raw --container-format=bare --is-public=true < cirros.raw
```

③ 在 Ceph 的镜像池中查询镜像。开发者可以在 Ceph 的镜像池中查询镜像 ID 来验证新添加的镜像。

```
# rados -p rbd ls --name client.rbd --keyring /etc/ceph/ceph.client.rbd.keyring | grep -i id
rbd_id.34f4fc11-74b0-42c0-8080-c7a4209ec28e
```

现在已经将 Glance 的默认存储后端配置改为 Ceph，所有的 Glance 镜像都将存储在 Ceph 中。

3. 配置 Nova 服务

为了将所有 OpenStack 实例放进 Ceph，需要为 Nova 配置一个临时性的后端。要做到这一点，须在 OpenStack 的计算节点（即 Training 节点）上修改/etc/nova/nova.conf 配置文件。

（1）修改服务配置文件

修改 nova.conf 配置文件中［libvirt］部分并添加以下代码。

```
rbd_pool = rbd
rbd_ceph_conf = /etc/ceph/ceph. conf
rbd_flatten_volume_from_snapshot = false
rbd_max_clone_depth = 5
rbd_store_chunk_size = 4
rados_connect_timeout = -1
glance_api_version = 2
rbd_user = rbd
rbd_secret_uuid = 207a92a6-acaf-47c2-9556-e560a79ba472
```

（2）重启服务

重新启动 OpenStack Nova 服务。

```
# service openstack-nova-compute restart
```

（3）配置 Libvirt

在 OpenStack 的计算节点上生成 UUID，定义 secret. xml 文件，设置密钥给 Libvirt，这里在 Training 节点上进行操作。

① 使用如下代码生成 UUID。

```
[ root@ Training ~ ]# uuidgen
207a92a6-acaf-47c2-9556-e560a79ba472
```

② 创建密钥文件，并将 UUID 设置给它。

```
cat > secret. xml <<EOF
<secret ephemeral = 'no' private = 'no' >
<uuid>207a92a6-acaf-47c2-9556-e560a79ba472</uuid>
<usage type = 'ceph' >
<name>client. rbd secret </name>
</usage>
</secret>
EOF
```

③ 定义（define）密钥文件，并保证生成的保密字符串是安全的。在接下来的步骤中需要使用这个保密的字符串值。

```
# virsh secret-define --file secret. xml
Secret 207a92a6-acaf-47c2-9556-e560a79ba472 created
```

④ 在 virsh 里设置好最后一步生成的保密字符串值，创建完成后查看系统的密钥文件。

```
# virsh secret-set-value --secret 207a92a6-acaf-47c2-9556-e560a79ba472 --
base64 $ (cat client. rbd. key)
```

```
Secret value set
# virsh secret-list
UUID                                  Usage
--------------------------------------
207a92a6-acaf-47c2-9556-e560a79ba472 Unused
```

（4）测试 Ceph

① 创建虚拟机。

配置 Nova 使用 Ceph 后，创建一台实例，并查看状态信息，net-id 可以使用“neutron net-list”命令来查询。

```
# nova boot --flavor  small --image "CirrOS 0.3.4" --nic net-id=6d0f1df5-2e86-4938-93eb-ac5958c7dd62  ceph-vm1
+--------------------------------------+--------------------------------------+
| Property                             | Value                                |
+--------------------------------------+--------------------------------------+
| OS-DCF:diskConfig                    | MANUAL                               |
| OS-EXT-AZ:availability_zone          | nova                                 |
| OS-EXT-SRV-ATTR:host                 | -                                    |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | -                                    |
| OS-EXT-SRV-ATTR:instance_name        | instance-0000000c                    |
| OS-EXT-STS:power_state               | 0                                    |
| OS-EXT-STS:task_state                | scheduling                           |
| OS-EXT-STS:vm_state                  | building                             |
| OS-SRV-USG:launched_at               | -                                    |
| OS-SRV-USG:terminated_at             | -                                    |
| accessIPv4                           |                                      |
| accessIPv6                           |                                      |
| adminPass                            | fwEnnnkAE2m2                         |
| config_drive                         |                                      |
| created                              | 2017-07-17T12:55:50Z                 |
| flavor                               | small(6)                             |
| hostId                               |                                      |
| id                                   | 12a64f7f-794d-494a-8030-fb1df837c566 |
| image                                | CirrOS 0.3.4(34f4fc11-74b0-42c0-8080-c7a4209ec28e) |
| key_name                             | -                                    |
| metadata                             | {}                                   |
| name                                 | ceph-vm1                             |
```

```
| os-extended-volumes:volumes_attached  | []                                   |
| progress                              | 0                                    |
| security_groups                       | default                              |
| status                                | BUILD                                |
| tenant_id                             | 4645f1b850aa463389450757e6321b23     |
| updated                               | 2017-07-17T12:55:50Z                 |
| user_id                               | 907a508919ff49d8aea2a0b6afd68f82     |
+---------------------------------------+--------------------------------------+
```

② 检查结果。在 Ceph 存储群中检查实例信息。

```
# rados  -p rbd ls --name client.rbd --keyring /etc/ceph/ceph.client.rbd.keyring
| grep -i id
rbd_id.12a64f7f-794d-494a-8030-fb1df837c566
```

至此，在 Nova 中启用 Ceph 作为实例的存储后端已经测试完成，可以使用 Ceph 完成虚拟机的磁盘存储和操作系统的镜像存储。

项目实训

【实训题目】

配置 OpenStack 的 Nova 服务使用 Ceph 作为统一存储。

【实训目的】

1. 掌握 Ceph 客户端的安装方法。
2. 掌握 OpenStack 的 Nova 服务使用 Ceph 的方法。

【实训内容】

1. 配置 OpenStack Ceph 客户端。
2. 配置 OpenStack 的 Nova 服务并重启服务。
3. 配置 Libvirt。
4. 创建一台虚拟机，并验证。

单元小结

本单元介绍了 Ceph 的客户端的基本安装和配置、OpenStack 服务使用 Ceph 存储的方法。通过对服务的安装以及配置使用，提高了读者对 Ceph 存储的运用能力。通过本单元的学习，相信读者可以熟练地掌握 Ceph 的使用方法，对 Ceph 在不同场合的使用也有了自己清晰的认识。

参 考 文 献

[1] Singh K. Ceph Cookbook 中文版 [M]. Ceph 中国社区，KVM 云技术社区，译. 北京：电子工业出版社，2016.

[2] 查伟. 数据存储技术与实践 [M]. 北京：清华大学出版社，2016.

[3] 叶毓敏，等. 软件定义存储 [M]. 北京：机械工业出版社，2016.

[4] 武志学，赵阳，马超英. 云存储系统：Swift 的原理、架构及实践 [M]. 北京：人民邮电出版社，2015.

CETC 云计算技术与应用专业校企合作系列教材

◇ Android 云存储客户端开发

◇ 软件定义网络（SDN）技术与实践

◇ Docker 容器技术与应用

◆ 云存储技术与应用

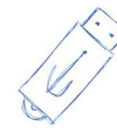
◇ Hadoop 大数据平台构建与应用

◇ 云计算数据中心运维

◇ 虚拟化技术与应用

◇ 云计算网络技术与应用

◇ Python 语言



ISBN 978-7-04-049103-6

9 787040 491036

定价 24.80 元